OFFSIDE LABS

Jupiter Lend Liquidity & Lending

Smart Contract Security Assessment

August 2025

Prepared for:

Jupiter

Prepared by:

Offside Labs

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 *Fluid Solana* smart contracts, starting on July 10th, 2025, and concluding on July 18th, 2025.

Project Overview

The Liquidity Layer of Fluid is a foundational component that consolidates liquidity across various protocols, enhancing capital efficiency, security, and user experience. It facilitates seamless interactions between protocols, allowing new innovations to be easily integrated while maintaining automated limits to safeguard protocol funds. Additionally, the layer supports diverse protocol designs and risk models, enabling versatile functionalities.

The lending protocol is a concise lend and earn protocol which provides direct access to Fluid's Liquidity Layer, enabling efficient and secure lending activities.

Audit Scope

The assessment scope contains mainly the smart contracts of the *lending* and *liquidity* program for the *Fluid Solana* project.

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

- *Fluid Solana:*
  - Codebase: `https://github.com/Instadapp/fluid-contracts-solana`
  - Branch: audit-2
  - Commit Hash: 2475151aa56cba12688d7cdd1f8f319aec797b07

We listed the files we have audited below:

- *Fluid Solana:*
  - programs/lending/src/*.rs
  - programs/liquidity/src/*.rs
  - programs/library/src/*.rs

Findings

The security audit revealed:

- 0 critical issue
- 0 high issue
- 5 medium issues
- 3 low issues
- 5 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	Precision Error in supply_ratio May Lead to DoS	Medium	Fixed
02	Token Reserve Should be Updated Before Config Updated	Medium	Fixed
03	Lack of Rate Update in UpdateRewardsRateModel IX	Medium	Fixed
04	Token Exchange Price Will be Incorrect When Update Time Range Crosses Reward Period Boundary	Medium	Fixed
05	Precision Loss and Inflation Attack	Medium	Partially Fixed
06	Incorrect Division Math Library	Low	Fixed
07	NotSet Status Should be Blocked in Operate IX	Low	Fixed
08	Different Token Programs of mint and f_token_mint May Lead to DoS	Low	Fixed
09	unpause_user IX Can Skip User Position Config NotSet Status	Informational	Fixed
10	Operate.protocol Can Not be Mutable	Informational	Fixed
11	Source Token Account of Rebalance Must be Pre-initialized	Informational	Fixed
12	Overly Restrictive MIN_TOKEN_DECIMALS	Informational	Acknowledged
13	CPI Error Can Not be Captured	Informational	Acknowledged

4 Key Findings and Recommendations

4.1 Precision Error in supply_ratio May Lead to DoS

Severity: Medium	Status: Fixed
Target: Liquidity Program	Category: Precision Issue

Description

The `get_supply_ratio` function has a precision of 1e4, so when `total_supply_with_interest` is less than 1 bps of `total_supply_interest_free`, the `supply_ratio` will return 0.

The issue is that, if `total_supply_with_interest` is not 0, the function `calculate_exchange_prices` calculates the reciprocal of the `supply_ratio`. Then it panics in this case.

```
            if self.total_supply_with_interest <
            ↪   self.total_supply_interest_free {
                // ratio is supplyWithInterest / supplyInterestFree
                ↪   (supplyInterestFree is bigger)

                let supply_ratio: u128 =
                ↪   EXCHANGE_PRICE_RATE_OUTPUT_DECIMALS
                    .safe_mul(FOUR_DECIMALS)?
                    .safe_div(supply_ratio)?;
```

programs/liquidity/src/state/token_reserve.rs#L427-L432

Impact

Once the above conditions are met, the entire protocol will cease all functionality due to the inability to update the latest exchange price. This scenario typically occurs accidentally during low-liquidity initialization, or could be exploited by malicious actors when there is no upper limit on `total_supply_interest_free`. An attacker could intentionally flood the protocol with excessive `total_supply_interest_free` liquidity to trigger the above DoS condition.

Recommendation

Optimize the calculation in `calculate_exchange_prices` by simplifying formulas and reducing division operations.

Mitigation Review Log

Fixed in commit 1ca22f9168fe9ee4be7ad0ac4acf33763f205efa.

4.2 Token Reserve Should be Updated Before Config Updated

Severity: Medium	Status: Fixed
Target: Liquidity Program	Category: Logic Error

Description

The current implementation includes multiple admin instructions for updating various configurations, which can affect the calculation of exchange prices. However, the update flow for some of these instructions is incorrect.

In the `update_token_config` instruction, the exchange prices (`supply_exchange_price` and `borrow_exchange_price`) are updated, but critical related values like `last_utilization` and `borrow_rate` are not updated, potentially leading to inconsistencies.

```
token_reserve.supply_exchange_price = supply_exchange_price;
token_reserve.borrow_exchange_price = borrow_exchange_price;
token_reserve.fee_on_interest = token_config.fee.cast()?;
token_reserve.max_utilization = token_config.max_utilization.cast()?;
token_reserve.last_update_timestamp =
↪   Clock::get()?.unix_timestamp.cast()?;
```

programs/liquidity/src/module/admin.rs#L319-L323

In the `update_user_supply_config` and `update_user_borrow_config` instructions, exchange prices are expected to be updated based on the amounts before the configuration changes (e.g., `raw_interest` and `interest_free` amounts). However, the current implementation updates the exchange prices after modifying these amounts. This results in one of the exchange prices being calculated incorrectly.

```
token_reserve.set_total_supply_with_interest(total_supply_raw_interest)?;
token_reserve.set_total_supply_interest_free(total_supply_interest_free)?;

let rate_model = context.accounts.rate_model.load()?;
// trigger update borrow rate, utilization, ratios etc.
token_reserve.update_exchange_prices_and_rates(&rate_model)?;
```

programs/liquidity/src/module/admin.rs#L543-L548

Impact

The incorrect flow for updating exchange prices can result in inaccurate exchange price calculations.

Recommendation

Update the exchange prices and rates by `update_exchange_prices_and_rates` before applying changes to the state.

Mitigation Review Log

Fixed in commit 630b78c05ba0ef8a201a79924e3ce0dd24bf8747 and 1ca22f9168fe9ee4be7ad0ac4acf33763f205efa.

4.3 Lack of Rate Update in UpdateRewardsRateModel IX

Severity: Medium	Status: Fixed
Target: Lending Program	Category: Logic Error

Description

In the `update_reward_rate_model` instruction, the implementation simply replaces the `rewards_rate_model` as shown below:

```
    let lending = &mut ctx.accounts.lending;
    lending.rewards_rate_model =
    ↪ ctx.accounts.new_rewards_rate_model.key();

    Ok(emit!(LogUpdateRewards {
        rewards_rate_model: ctx.accounts.new_rewards_rate_model.key(),
    }))
```

programs/lending/src/module/admin.rs#L115-L120

However, this implementation does not account for updating the reward rate if the old reward model has been active during the time since `last_update_timestamp` . This oversight could cause discrepancies in the reward calculation.

Impact

Rewards accrued during the time since `last_update_timestamp` may not be correctly distributed.

Recommendation

Consider the following improvements:

1. Force an update to the reward rate before changing the `rewards_rate_model` .
2. Add a parameter to the `update_reward_rate_model` instruction to determine whether the rate should be updated at the time of the model change.

Mitigation Review Log

Fluid Team: *We do not expect that the rewards_rate_model is ever changed after setting it for the first time.*

Fixed in commit d92e582b25db0cdafc227074246d0e862ef7c926.

4.4 Token Exchange Price Will be Incorrect When Update Time Range Crosses Reward Period Boundary

Severity: Medium	**Status: Fixed**
Target: Lending Program	**Category: Logic Error**

Description

The current implementation of the token exchange price calculation has a logic flaw when the time range since `last_update_timestamp` crosses a reward period boundary. Specifically, the issue lies in how the `rewards_rate` is calculated based on the current time and its reward period. The logic does not account for situations where the time range spans multiple reward periods, leading to incorrect calculations.

The relevant code snippet is as follows:

```rust
        let (mut rewards_rate, rewards_ended, rewards_start_time) =
        ↪   current_rate_model.get_rate(
        old_token_exchange_price
            .cast::<u128>()?
            .safe_mul(f_token_total_supply.cast()?)?
            .safe_div(EXCHANGE_PRICES_PRECISION.cast()?)?
            .cast()?,
    )?;

    if rewards_rate > MAX_REWARDS_RATE.cast()? || rewards_ended {
        // rewardsRate is capped, if it is bigger > MAX_REWARDS_RATE,
        ↪   then the rewardsRateModel
        // is configured wrongly (which should not be possible). Setting
        ↪   rewards to 0 in that case here.
        rewards_rate = 0;
    }
```

programs/lending/src/utils/helpers.rs#L116-L128

If the `last_update_timestamp` falls within one reward period while the current time falls in a subsequent reward period (or beyond the end time of any reward period), the entire duration between `last_update_timestamp` and the current time will incorrectly use the reward rate of the current period. This ignores the portion of time that should have been calculated using the previous reward rate.

Impact

When the time range since `last_update_timestamp` crosses a reward period boundary, the new token exchange price will be calculated using only the most recent reward rate.

Recommendation

The `get_rate` function should be modified to account for the `last_update_timestamp`.

Mitigation Review Log

Fixed in PR39 and PR42.

4.5 Precision Loss and Inflation Attack

Severity: Medium | **Status: Partially Fixed**

Target: Liquidity and Lending Program | **Category: Precision Issue**

Description

When supply/borrow exchange price of the liquidity program is not 1(EXCHANGE_-PRICES_PRECISION), or token exchange price of lending program `f_token_mint` is not 1(due to rewards/interest accumulation), an attacker can use the precision loss in the deposit/withdraw calculation to amplify the exchange price exponentially, or introduce inconsistencies in accounting between the lending program and liquidity program.

Impact

1. Precision loss in deposit and withdraw IXs of lending program:
   - `programs/lending/src/utils/deposit.rs#L70-L74`
   - `programs/lending/src/utils/withdraw.rs#L57-L61`

   For example, if the current `token_exchange_price` is 3, and a user attempts to deposit 5 assets via the deposit instruction, precision loss during the calculation of `shares_minted` results in only 1 share being issued instead. This causes the user to suffer precision loss exceeding 1 asset.
2. Precision loss in operate instruction of liquidity program:
   Since the operate instruction also calculates `new_supply_interest_raw` using only the input asset quantity, it suffers from the same precision loss issue.
3. Risk of inflation attack: When a pool is newly initialized in the liquidity program with zero liquidity, the first depositor can exploit these two precision loss issues to donate assets to the pool, artificially inflating the exchange price exponentially. In this scenario, all subsequent depositors and borrowers will suffer significant precision loss during operations, with most of these losses effectively becoming profits for the attacker.

4. Accounting inconsistencies caused by precision loss will expose the lending protocol to deficits risk:
   Reward emissions cause the lending program's `token_exchange_price` to outpace the liquidity program's `supply_exchange_price` , leading to asymmetric precision loss where the protocol absorbs more losses than users.
   For example, the `token_exchange_price` of is 4, and `supply_exchange_price` is 3, a user deposits 8 assets into lending program. During deposit instruction, lending program mints 2 shares of f_token_mint, but due to precision loss, only 2 of `new_supply_interest_raw` is added to the supply position of the lending protocol in the liquidity program. At the same slot, the user withdraws the 2 shares from lending program. The lending program needs to withdraw 3 `new_supply_interest_raw` of the supply position from the liquidity program to return 8 assets to the user. This will result in a protocol loss of 3 assets(1 `supply_interest_raw`).

Recommendation

1. After each calculation of shares from assets, perform a reverse calculation to determine the minimum required assets. Transfer only this minimal amount of assets.
2. Optional: Preemptively mint dead shares (e.g., 1000) during pool initialization to mitigate precision-based attacks.
3. Optional: Ensure precision loss during asset conversions between multi prices always falls on users to maximally protect protocol solvency.

Mitigation Review Log

Fluid Team: Added explicit revert if new amount raw ends up 0 for the impact 2. here: PR-55

Inflating exchange price of the impact 4. should not be possible especially not after the above additional check. Also, for all our protocols we always create the first position which stays there permanently do resolve all these kind of possible scenarios.

Putting the other impacts in our lower priority hardenings task list to think about deeper later on for now.

4.6 Incorrect Division Math Library

Severity: Low	Status: Fixed
Target: Library	Category: Math Error

Description

The math library introduces two division methods: `ceil_div` and `floor_div` . However, the implementation overlooks whether the quotient is positive or negative, which results in incorrect calculations in certain cases.

```rust
    fn checked_ceil_div(&self, rhs: $t) -> Option<$t> {
      let quotient = self.checked_div(rhs)?;

      let remainder = self.checked_rem(rhs)?;

      if remainder > <$t>::zero() {
          quotient.checked_add(<$t>::one())
      } else {
          Some(quotient)
      }
  }
```

programs/library/src/math/ceil_div.rs#L17-L27

For the input `8.checked_ceil_div(-3)` , the expected result is `-2` , but the function incorrectly calculates `-1` .

As a baseline, the following tests in examples only have 50% pass rate.

https://doc.rust-lang.org/std/primitive.i32.html#method.div_ceil

https://doc.rust-lang.org/std/primitive.i32.html#method.div_floor

Impact

This flawed division implementation can lead to incorrect calculations. While the current codebase may not trigger these cases, the issue leaves room for potential future bugs and logic errors, especially when extended to other contexts.

Recommendation

Update the implementation to consider the sign of the quotient during the calculation.

Mitigation Review Log

Fixed in commit 9bea6051f8923799129eb7c0664625b87a7a837d and 844bd43d662130c58bc60922518d5d9895a83264.

4.7 NotSet Status Should be Blocked in Operate IX

Severity: Low	Status: Fixed
Target: Liquidity Program	Category: Logic Error

Description

The `operate` instruction only checks if the user supply/borrow positions are paused. It should also check if the user positions are in `NotSet` status.

Recommendation

If the configs of user positions are not set, the operation should be blocked.

Mitigation Review Log

Fixed in commit 6c3b46d3a1a23b075396b13ca5420313748e72aa.

4.8 Different Token Programs of mint and f_token_mint May Lead to DoS

Severity: Low	**Status: Fixed**
Target: Lending Program	**Category: DoS risk**

Description

In the `init_lending` instruction, there is no restriction ensuring that the `f_token_mint` uses the same token program as `mint` .

```rust
    #[account(mut)]
    pub mint: InterfaceAccount<'info, Mint>,

    #[account(
        init,
        seeds = [F_TOKEN_MINT_SEED, mint.key().as_ref()],
        bump,
        payer = signer,
        mint::decimals = mint.decimals,
        mint::authority = lending_admin,
        mint::token_program = token_program,
    )]
    pub f_token_mint: Box<InterfaceAccount<'info, Mint>>,
```

programs/lending/src/state/context.rs#L51-L63

During initialization, the `f_token_mint` can be created using a different token program than `mint` . However, in subsequent user instructions, both `mint` and `f_token_mint` are expected to use the same token program.

Impact

If the `f_token_mint` was initialized with a different token program, operations such as `mint` , `burn` , or `transfer` will fail due to an ownership mismatch between the token programs.

Recommendation

Consider the following solutions:

1. Ensure `f_token_mint` uses the same token program as `mint` .
2. Pass two token programs in user instructions

Mitigation Review Log

Fixed in commit 56ccd1d5fabfc858444625ec1e26eb0d2d7a12c7.

4.9 Informational and Undetermined Issues

unpause_user IX Can Skip User Position Config NotSet Status

Severity: Informational	**Status: Fixed**
Target: Liquidity Program	**Category: Logic Error**

If a user position of `UserSupplyPositionStatus::NotSet` status is paused, when calling the `unpause_user` instruction, the status will be set to `Active` without setting config.

programs/liquidity/src/module/admin.rs#L747-L748

It's recommended to check the status is `UserSupplyPositionStatus::Active` when pausing a user position in the `pause_user` instruction.

Operate.protocol Can Not be Mutable

Severity: Informational	**Status: Fixed**
Target: Liquidity Program	**Category: Data Validation**

The lending program uses the `lending` account as the `protocol` account for liquidity program. Since the lending account is an initialized PDA assigned to the lending program, this account cannot be modified by any CPI other than the lending program itself. Therefore, the `mut` constraint on the `protocol` account in the `operate` instruction of the liquidity program should be removed.

Source Token Account of Rebalance Must be Pre-initialized

Severity: Informational	Status: Fixed
Target: Lending Program	Category: Data Validation

The `depositor_token_account` of `Rebalance` ix is the source token account of the `assets_delta` transaction. It must be initialized before calling the instruction if `assets_delta` is not zero.

So the `init_if_needed` constraint here is improper:

```
    #[account(
        init_if_needed,
        payer = signer,
        associated_token::mint = mint,
        associated_token::authority = signer,
        associated_token::token_program = token_program
    )]
    pub depositor_token_account: Box<InterfaceAccount<'info,
     ↪  TokenAccount>>,
```

programs/lending/src/state/context.rs#L448-L455

Overly Restrictive MIN_TOKEN_DECIMALS

Severity: Informational	Status: Acknowledged
Target: Liquidity Program	Category: Logic Error

The current `MIN_TOKEN_DECIMALS` is 6. But many mainstream meme coins commonly have fewer than 6 decimals. For example, Bonk has decimals of 5: https://solscan.io/token/DezXAZ8z7PnrnRJjz3wXBoRgixCa6xjnB7YaB1pPB263

CPI Error Can Not be Captured

Severity: Informational	Status: Acknowledged
Target: Lending Program	Category: Logic Error

Several code snippets attempt to handle CPI errors, but these errors are not propagated and cannot be captured during SVM execution. An example of such a code snippet is as follows:

programs/lending/src/invokes/liquidity_layer.rs#L52-L59

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