OFFSIDE LABS

Jupiter Perpetuals Exchange

Smart Contract Security Assessment

May 2025

Prepared for:

Jupiter

Prepared by:

Offside Labs

Ronny Xing

Yao Li

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 Perpetuals* smart contracts, starting on April 8, 2025, and concluding on April 10, 2025.

Project Overview

Jupiter Perpetuals exchange is a novel LP-to-trader perpetual exchange on Solana, offering up to 100x leverage. Utilizing LP pool liquidity and oracles, it ensures zero price impact, zero slippage, and deep liquidity. Oracles enable stable market operations during liquidations and stop-loss events, removing risks of position bankruptcy and LP pool fund loss.

JLP's native lending feature enables users to deposit JLP tokens as collateral to borrow underlying assets (USDC, BTC, SOL, ETH).

Jupiter Perpetuals' high-reliability price oracles and native JLP burn/mint mechanism eliminate liquidity challenges inherent to traditional liquidation models. This feature maximizes underlying asset utilization while minimizing risk, with compound interest generated from borrowing directly accruing to JLP's value.

Audit Scope

The assessment scope contains mainly the smart contracts of the *Perpetuals* program for the *Jupiter Perpetuals Exchange* project.

- Codebase: PR-287
- Commit Hash: a4012452e64e41c11818ce85f811ad52878764d8

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

- *Jupiter Perpetuals*
  - programs/perpetuals/src/*.rs

Findings

The security audit revealed:

- 0 critical issue
- 0 high issue
- 2 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	Excessive Precision Loss in BorrowPosition.borrow_size	Medium	Fixed
02	Avoid Using Configuration Toggles in State Functions	Medium	Fixed
03	Ambiguity Between owned and theoretically_owned	Low	Fixed
04	Suggest Independent Interest Accrual and Delayed Accounting	Low	Fixed
05	Compound Interest Formula Failure Due to Precision Loss	Low	Fixed
06	Event Fields are Incorrect	Informational	Fixed
07	Lack of Fee Config Validation in BorrowLendParams	Informational	Fixed
08	Typo of borrows_funding_rate_state Initialization	Informational	Fixed
09	Suggest to Consolidate Equivalent Branches	Informational	Fixed
10	Edge Case May Break Borrow Interest Updating for AUM	Informational	Acknowledged

4 Key Findings and Recommendations

4.1 Excessive Precision Loss in BorrowPosition.borrow_size

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

Description

The `get_interests_accumulated` method's use of rounding-up in interest calculations causes inflationary distortion in `BorrowPosition.borrow_size`.

```
653         let interests = math::checked_as_u64(math::checked_ceil_div(
654             math::checked_mul(
655                 math::checked_sub(
656                     compounded_interest_index,
657 
    ↪ borrow_position.cumulative_compounded_interest_snapshot,
658                 )?,
659                 total_borrows as u128,
660             )?,
661             borrow_position.cumulative_compounded_interest_snapshot,
662         )?)?;
```

programs/perpetuals/src/state/custody.rs#L653-L662

Impact

This precision error compounds with each update, resulting in material accuracy loss for users with frequently refreshed positions.

Recommendation

Should use u128 and scaled decimals to calculate and store `BorrowPosition.borrow_size`. Apply ceiling rounding only for token amount calculation in repay / liqudation / or even every borrow ratio calculation and comparison.

Please note that, for repay or liquidation operations which decrease the global `custody.debt`, its always better to ceiling round the debt reduced, because due to precision error, the sum of borrow sizes of all borrow positions will be less than the global `custody.debt`. Applying ceiling-rounding to debt elimination operations will neutralize this precision error, ensuring neither global debt nor AUM are overstated.

Mitigation Review Log

Fixed in commit b53f0746be018df500850b95f971afbb3cb1868e.

4.2 Avoid Using Configuration Toggles in State Functions

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

Description

Function `enable_borrows` uses `borrows_limit_in_bps > 0` as a config flag to determine whether this custody is lendable. This function is also used in some internal state functions or validation functions at the following locations:

- programs/perpetuals/src/state/custody.rs#L265-L265
- programs/perpetuals/src/state/custody.rs#L493-L493
- programs/perpetuals/src/state/custody.rs#L699-L699

Impact

When disabling lending support for a custody that currently carries debt, the above conditional branch logic would cause accounting inconsistencies. While the debt would cease to be accounted upon lending deactivation, these obligations remain valid and should continue accruing interest at the appropriate rate until either repayment or liquidation occurs.

Recommendation

Avoid using `enable_borrows` as a conditional branch in internal state methods. This flag should exclusively gatekeep the borrow instruction.

Mitigation Review Log

Fixed in commit 1c2742c82def4608bdacbb03d12585f19536bc29.

4.3 Ambiguity Between owned and theoretically_owned

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

Description

The following branch conditions for `self.assets.owned == TokenAmount(0)` should use `theoretically_owned()` instead of `assets.owned`.

- programs/perpetuals/src/state/custody.rs#L707-L707
- programs/perpetuals/src/state/custody.rs#L864-L864

Recommendation

Use `theoretically_owned()` .

Mitigation Review Log

Fixed in PR-287 .

4.4 Suggest Independent Interest Accrual and Delayed Accounting

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

Description

There are two material changes implemented after design V0:

- The pool supports multiple borrowable assets;
- Interest and protocol fees will be collectively distributed and tracked via the WithdrawFees2 instruction.

They lead to an accounting issue for AUM:

The real-time debt accrual into `theoretically_owned` creates AUM calculation inaccuracies when `get_assets_under_management_usd_with_doves()` is called prior to `custody.update_borrow_position()` . With multiple borrowing enabled custodies in the pool, precise AUM calculation require expensive CU consumption - as each update needs to update funding rates and accrued interest across all lending active custodies.

Recommendation

Therefore, we recommend decoupling accrued interest from global debt accounting, and merge them in the WithdrawFees2 ix for delayed AUM updates.

Detailed Modification:

- Change function `Custody.get_debt()` to `self.debt - self.borrow_lend_interests_accured` ;
- Change function `Custody.theoretically_owned()` to `self.assets.owned + self.get_debt()` , dont need to sub `protocol_fee` anymore.

This modification additionally reduces CU cost for each `theoretically_owned()` calculation.

Mitigation Review Log

Fixed in commit b75729b98e8f8c67e5ef03aef1d8ea860b239bf0.

4.5 Compound Interest Formula Failure Due to Precision Loss

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

Description

The function `approximate_compounded_interest` uses an approximate formula to calculate the compound interest. The issue is that, the interest rate and the expansions of the formula maintain only 12 decimal places of precision, due to `COMPOUNDING_RATE_POWER` is 1e12.

To activate the second expansion (i.e., ensure the second expansion is greater than 0), the minimum `interest_rate_per_second` must be equal to or greater than 1e6 if the `fraction_one` is 1e12. It means the minimum APR is `1e6 * 3600 * 24 * 365 / 1e12 * 100% = 3153.6%` . This is an impossible parameter in a normal lending protocols.

Proof of Concept

There is a PoC test to dump all of the expansions of the approximate formula when the APR is >100%.

Output:

```
running 1 test
2739744000 0 0 0
19178208000 0 0 0
82192320000 0 0 0
1000006560000 0 0 0
thread 'state::custody::tests::test_compounding_approximation2' panicked
 ↪  at
'days 365, approx 2000006560000, float 2718299616766'
```

Recommendation

1. Remove the fourth expansion. For per-second compound interest, the fourth term of the approximation series becomes irrelevant because:
    a. The first three terms already provide sufficient accuracy for loan durations up to 5 years;
    b. Standard decimal precision (typically 18-27 digits) renders the fourth term numerically zero.
2. Compound interest calculations and representations must use at least 18 decimal places of precision.
3. Update the unit test `state::custody::tests::test_compounding_approximation` , because it uses the `Perpetuals::RATE_POWER` as the `fraction_one` instead of the

current decimal precision(might be updated after the mitigation). And it uses impossible test cases with an `interest_rate_per_second` of 10%(APR 315360000%)

Mitigation Review Log

Fixed in commit 0cbc87e3d7ce70536aad6f154e825cdf1ff10795.

4.6 Informational and Undetermined Issues

Event Fields are Incorrect

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

1. Use token amount as usd vaule:
    a. programs/perpetuals/src/instructions/repay_to_custody.rs#L127-L127
    b. programs/perpetuals/src/instructions/borrow_from_custody.rs#L189-L189
    c. programs/perpetuals/src/instructions/withdraw_collateral_for_borrows.rs#L202-L202
2. Use old collateral_amount as new collateral_amount: programs/perpetuals/src/instructions/withdraw_collateral_for_borrows.rs#L201-L201

Lack of Fee Config Validation in BorrowLendParams

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

Recommend to add fee config validation for `protocol_fee_bps` and `liquidation_fee_bps` in the function `BorrowLendParams.validate` .

The current max protocol fee in the pool config validation is `MAX_PROTOCOL_SHARE_BPS = 5_000` ;

And for `liquidation_fee_bps` , should add addtional validation to ensure

$\frac{1}{1+liquidation\_fee\_bps} > liquidation\_margin$

This check avoids any dead band of the liquidation fee bps.

Typo of borrows_funding_rate_state Initialization

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

```
141     custody.funding_rate_state.cumulative_interest_rate =
        Perpetuals::INITIAL_COMPOUNDING_INDEX;
142     custody.borrows_funding_rate_state.hourly_funding_dbps = 0;
```

programs/perpetuals/src/instructions/add_custody.rs#L141-L142

There is a typo in borrows_funding_rate_state initialization, `custody.funding_rate_state.cumulative_interest_rate` does not need an init value. Although there is no impact.

Suggest to Consolidate Equivalent Branches

Severity: Informational	Status: Fixed
Target: Optimization	Category: Logic Error

```
548 } else {
549     self.update_cumulative_funding_rate(current_time)?;
550     borrow_position.update_time = current_time;
551     borrow_position.cumulative_compounded_interest_snapshot =
552         self.borrows_funding_rate_state.cumulative_interest_rate;
553 }
```

programs/perpetuals/src/state/custody.rs#L548-L555

Suggest to remove the *else* branch, when `borrow_size` is zero, the logic in `update_borrow_position` is functionally equivalent to the *else* branch and incurs no CU.

Edge Case May Break Borrow Interest Updating for AUM

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

The current implementation does not update the borrow interest for AUM in real time. When the `withdraw_fee2` function is called, the borrow interest is incorporated into the AUM.

However, if `self.debt` falls below `self.borrow_lend_interests_accured`, the real-time AUM calculation will incorrectly include the interest due to an underflow scenario. This edge case violates the intended logic and assumptions of the system. The impact of this issue is limited but noteworthy.

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.

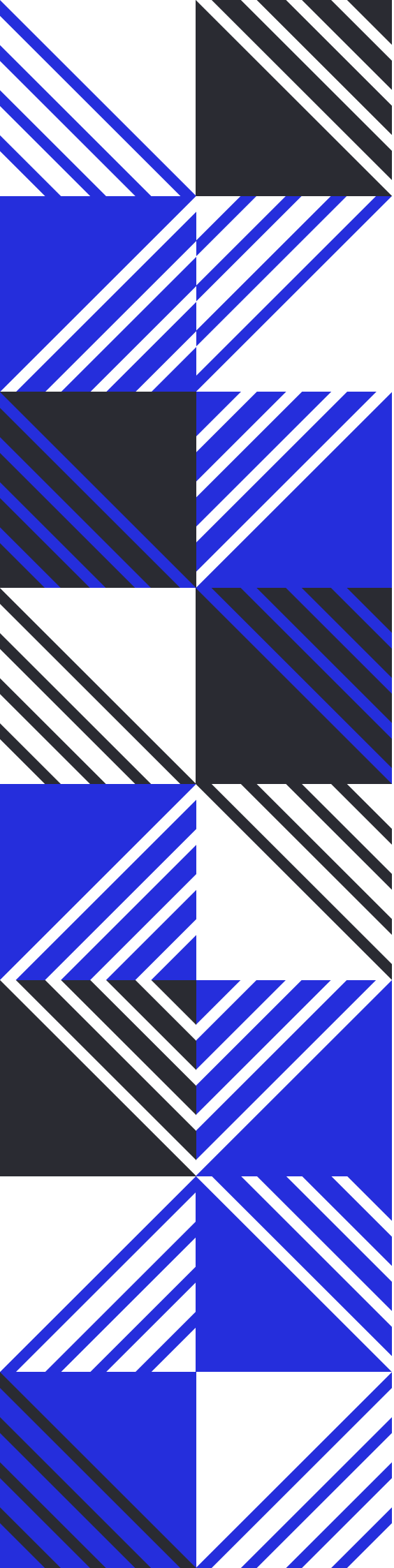
We expressly disclaim any liability for any losses or damages arising from the use of this report or from any security breaches that may occur in the future. We also recommend that our clients engage in multiple independent audits and establish a public bug bounty program as additional measures to bolster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