Three Sigma

Code Audit

OSTIUM

Ostium Onchain perpetuals for Real World Assets

Disclaimer

Code Audit

Ostium Onchain perpetuals for Real World Assets

Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgment of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code's manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

Table of Contents

Code Audit

Ostium Onchain perpetuals for Real World Assets

Table of Contents

Summary

Code Audit

Ostium Onchain perpetuals for Real World Assets

Summary

Three Sigma audited Ostium in a 1.2 person week engagement. The audit was conducted from 20–12-2024 to 25-12-2024.

Protocol Description

Ostium is the first decentralized perpetual exchange purpose-engineered for Real World Assets. A stablecoin-settled trading engine, dynamic fee structure, and proprietary in-house oracle enables leveraged, synthetic trading on a wide range of traditional market assets – from oil to natural gas, soybeans, and more – all fully onchain.

Scope

Code Audit

Ostium Onchain perpetuals for Real World Assets

Scope

Filepath	nSLOC
src/abstract/Delegatable.sol	40
src/OstiumTradingCallbacks.sol	452
src/OstiumTrading.sol	501
src/OstiumPriceUpKeep.sol	137
src/OstiumTradingStorage.sol	368
src/OstiumPairInfos.sol	512
Total	2010

Diff:
pull/66/files#diff-c9fc4b1f80f893b2c28d7fb8ca59d2da10912592c4fdd3e55b6bcddd23eea79f

Assumptions

External dependencies are considered secure.

Methodology

Code Audit

Ostium Onchain perpetuals for Real World Assets

Methodology

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

Taxonomy

In this audit we report our findings using as a guideline Immunefi's vulnerability taxonomy, which can be found at immunefi.com/severity-updated/. The final classification takes into account the severity, according to the previous link, and likelihood of the exploit. The following table summarizes the general expected classification according to severity and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

Severity / Likelihood	LOW	MEDIUM	HIGH
NONE	None		
LOW	Low		
MEDIUM	Low	Medium	Medium
HIGH	Medium	High	High
CRITICAL	High	Critical	Critical

Project Dashboard

Code Audit

Ostium Onchain perpetuals for Real World Assets

Project Dashboard

Application Summary

Name	Ostium
Commit	97059f34415b3649cb7996483560e5372ec789e2
Language	Solidity
Platform	Arbitrum

Engagement Summary

Timeline	20-12-2024 to 25-12-2024
№ of Auditors	2
Review Time	1.2 person weeks

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	0	0	0
High	1	1	0
Medium	1	1	0
Low	1	1	0

None	0	0	0

Category Breakdown

Suggestion	0
Documentation	0
Bug	3
Optimization	0
Good Code Practices	0

Code Maturity Evaluation

Code Audit

Ostium Onchain perpetuals for Real World Assets

Code Maturity Evaluation

Code Maturity Evaluation Guidelines

Category	Evaluation
Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system.
Arithmetic	The proper use of mathematical operations and semantics.
Centralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Code Stability	The extent to which the code was altered during the audit.
Upgradeability	The presence of parameterizations of the system that allow modifications after deployment.
Function Composition	The functions are generally small and have clear purposes.
Front-Running	The system's resistance to front-running attacks.
Monitoring	All operations that change the state of the system emit events, making it simple to monitor the state of the system. These events need to be correctly emitted.
Specification	The presence of comprehensive and readable codebase documentation.
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage.

Code Maturity Evaluation Results

Category	Evaluation
Access Controls	Satisfactory. The codebase has a strong access control mechanism.
Arithmetic	Satisfactory. The codebase uses Solidity version >0.8.0 implementing safe arithmetic.
Centralization	Moderate. The admins/owners of the protocol have some privileges over the protocol (e.g setting parameters or oracle prices).
Code Stability	Satisfactory. The codebase was stable throughout the audit.
Upgradeability	Satisfactory. Most contracts are set up to allow for upgradability.
Function Composition	Satisfactory. Functionalities are well split into different contracts and helpers.
Front-Running	Satisfactory. No significant front-running issues were found.
Monitoring	Satisfactory. Most state changing occurrences emit events.
Specification	Satisfactory. The code matched the design specifications.
Testing and Verification	Satisfactory. Unit and fuzz tests were present for most functionality.

Findings

Code Audit

Ostium Onchain perpetuals for Real World Assets

Findings

3S-Ostium-H01

`OstiumTrading::removeCollateral()` can be called twice in a row to obtain huge leverage values

Id	3S-Ostium-H01
Classification	High
Severity	High
Likelihood	High
Category	Bug
Status	Addressed in #703ca32.

Description

`OstiumTrading::removeCollateral()` does not check if there are pending remove collateral, nor **`OstiumTradingCallbacks::handleRemoveCollateral()`** checks if the resulting leverage is bigger than the maximum, allowing users to call the function twice in a row to increase their leverage to huge values. This allows users to remove their whole collateral while keeping the same exposure, having very little risk.

As can be seen in the profit calculation below, the increased leverage will mean the huge gets a huge profit, getting risk free trades.

```
    function currentPercentProfit(
        ...
    ) private pure returns (int256 p) {
        int256 maxPnlP = int16(MAX_GAIN_P) * int32(PRECISION_6);
        p = (buy ? currentPrice - openPrice : openPrice - currentPrice) *
int32(PRECISION_6) * initialLeverage
            / openPrice;
        p = p > maxPnlP ? maxPnlP : p;
        p = p * leverage / initialLeverage;
    }
```

Recommendation

Either check if there are pending stored remove collateral orders or check the leverage when fulfilling the order.

3S-Ostium-M01

Collateral Calculation Rounding Down Allows Persistence of Dust Trades

Id	3S-Ostium-M01
Classification	Medium
Severity	Medium
Likelihood	Medium
Category	Bug
Status	Addressed in #36f7165.

Description:

The **`OstiumTrading`** smart contract exhibits an issue in its collateral calculation mechanism during the trade closure process, specifically within the **`closeTradeMarket()`** function of the **`OstiumTrading`** contract and the corresponding **`closeTradeMarketCallback()`** function in the **`OstiumTradingCallbacks`** contract.

This issue arises from the way collateral is calculated and rounded, leading to unintended bypassing of essential leverage checks.

In the **`OstiumTrading::closeTradeMarket()`** function, the calculation of **`remainingCollateral`** is performed using the following line of code:

```
uint256 remainingCollateral = t.collateral * (100e2 - closePercentage) / 100e2;
```

When a trade is initiated with a **`closePercentage`** near to 100%, for example, 99.99% (represented as 9999 in the contract) and a collateral amount of for example 5,500 units, the computation proceeds as follows:

```
remainingCollateral = 5500 * (10000 - 9999) / 10000 = 5500 * 1 / 10000 = 0
```

Due to Solidity's integer division, **`remainingCollateral`** truncates to zero. This outcome erroneously bypasses the subsequent conditional check intended to ensure that the remaining collateral maintains a minimum leveraged position:

```
if (
            remainingCollateral > 0
                && remainingCollateral * t.leverage / 100
                    <
IOstiumPairsStorage(registry.getContractAddress('pairsStorage')).pairMinLevP
os(pairIndex)
        ) {
            revert BelowMinLevPos();
        }
```

Since **remainingCollateral** is zero, the condition evaluates to false, allowing the trade to proceed without satisfying the minimum leverage requirement. This flaw is further exacerbated in the **OstiumTradingCallbacks::closeTradeMarketCallback()** function, where **collateralToClose** is calculated as follows:

```
uint256 collateralToClose = t.collateral * closePercentage / 100e2;
// Substituting the example values:
collateralToClose = 5500 * 9999 / 10000 = 5499
```

This results in a remainingCollateral of 1 unit (i.e., 5,500 - 5,499), effectively creating a dust trade. Although such minimal collateral may appear inconsequential and unlikely to pose significant operational risks, it disrupts the contract's invariant by leaving a position smaller than the mandated minimum. Over time, the accumulation of these dust trades could undermine the system's integrity and reliability, potentially leading to unforeseen vulnerabilities and inconsistencies within the trading platform.

Recommendation:

To address this vulnerability, it is recommended to modify the conditional check within the **OstiumTrading::closeTradeMarket()** function. The condition should be adjusted to ensure that the **remainingCollateral** is only subjected to the minimum leveraged position check if the **closePercentage** is not equal to 100%. This adjustment prevents scenarios where a high closePercentage, such as 99.99%, inadvertently results in a zero **remainingCollateral**, thereby maintaining the contract's integrity.

The revised condition can be implemented as follows:

```
if (
    closePercentage != 100e2
        && remainingCollateral * t.leverage / 100
            < pairMinLevPos(pairIndex)
) {
    revert BelowMinLevPos();
}
```

3S-Ostium-L01

Max **`pnl`** in **`OstiumTradingCallbacks::currentPercentProfit()`** can be surpassed

Id	3S-Ostium-L01
Classification	Low
Severity	Low
Likelihood	High
Category	Bug
Status	Addressed in #f69b5e0 and #b840a0f.

Description

The profit calculation is bound to **`maxPnlP = 900`** or '10x'. This used to work correctly because it was only possible to decrease the leverage. However, it is now possible to increase the leverage, which will allow the profit to exceed '10x'.

Recommendation

Check the maximum profit after applying the leverage multiplication.