Three Sigma Labs

Code Audit

NFTPerp Perpetual futures dex for NFTs

Disclaimer

Code Audit

NFTPerp Perpetual futures dex for NFTs

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

NFTPerp Perpetual futures dex for NFTs

Table of Contents

Summary

Code Audit

NFTPerp Perpetual futures dex for NFTs

Summary

Three Sigma Labs audited NFTPerp in an 8 person week engagement. The audit was conducted from 02-01-2024 to 26-01-2024.

Protocol Description

NFTPerp is an open-source DeFi platform enabling users to trade on the price of NFT collections like Milady and Pudgy Penguins using perpetual futures contracts, with all transactions in ETH.

Scope

Code Audit

NFTPerp Perpetual futures dex for NFTs

Scope

All smart-contracts present in the 'src' folder:

```
— src
    ├── AMM.sol
    ├── ClearingHouse.sol
    ├── ClearingHouseBase.sol
    ├── InsuranceFund.sol
    ├── PriceFeed.sol
    ├── clearinghouse-libs
    |   ├── Deleverager.sol
    |   ├── OrderCreator.sol
    |   ├── PositionManager.sol
    |   └── PositionMerger.sol
    ├── math
    |   ├── BitMath.sol
    |   ├── NFTPMath.sol
    |   └── TickMath.sol
    ├── nftperp-types
    |   ├── NFTPEnums.sol
    |   ├── NFTPErrors.sol
    |   └── NFTPStructs.sol
    └── utils
        ├── AccessControl.sol
        ├── AccessControlCH.sol
        └── TickBitmap.sol
```

Assumptions

The scope of the audit was carefully defined to include the contracts at the lowest level of the inheritance hierarchy, as these are the ones that will be deployed to the mainnet. External libraries from openZeppelin and Solady were used in the implementation of the contracts, but these libraries have already been audited and battle-tested by multiple protocols, guaranteeing a high level of security.

Methodology

Code Audit

NFTPerp Perpetual futures dex for NFTs

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

NFTPerp Perpetual futures dex for NFTs

Project Dashboard

Application Summary

Name	NFTPerp
Commit	1c16f2010a851ac23aec73822d55388901bbe5e7
Language	Solidity
Platform	Arbitrum

Engagement Summary

Timeline	02-01-2024 to 26-01-2024
№ of Auditors	2
Review Time	8 person weeks

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	1	1	0
High	1	1	0
Medium	2	2	0
Low	7	5	2
None	9	4	5

Category Breakdown

Suggestion	3
Documentation	0
Bug	12
Optimization	1
Good Code Practices	4

Code Maturity Evaluation

Code Audit

NFTPerp Perpetual futures dex for NFTs

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	Weak. The codebase was frequently updated throughout the audit.
Upgradeability	Satisfactory. Most contracts are setup up to allow for upgradability.
Function Composition	Satisfactory. Functionalities are well split into different contracts and helpers.
Front-Running	Moderate. Some front-running issues were identified.
Monitoring	Moderate. Some events should be added for state changing occurrences.
Specification	Satisfactory. The code matched the design specifications.
Testing and Verification	Moderate. Unit tests were present for most functionality but fuzzing and invariant tests could be performed.

Findings

Code Audit

NFTPerp Perpetual futures dex for NFTs

Findings

3S-NFTPerp-C01

`reduceOnly` limit order update does not take into account that a new amm may be selected, which may lead to loss of funds

Id	3S-NFTPerp-C01
Classification	Critical
Severity	Critical
Likelihood	Medium
Category	Bug
Status	Addressed in #6424621

Description

On a limit order update, in _updateLimitOrder(), changing amm is not being taken into account. The following scenarios are possible:

- if **`reduceOnly`** remains false, no problem

- if **`reduceOnly`** goes from false to true, no problem

- if **`reduceOnly`** remains true, it won't link the reduceOnly order in the new amm B, nor remove it from amm A

- if **`reduceOnly`** goes from true to false, it will remove the linked order from the new amm B, instead of the old amm A. This seems the worst scenario, as updating the limit order from false to true will pull weth from the trader, but this order may be deleted if the trader closes its position on amm A, losing the funds.

Recommendation

Correctly handle the case when the amm changes.

3S-NFTPerp-H01

Realized **`pnl`** not returned to trader on **`_mergeDecrease()`**

Id	3S-NFTPerp-H01
Classification	High
Severity	High
Likelihood	High
Category	Bug
Status	Addressed in the AMMRouter

Description

On the merge of a liquidator position following a liquidation, the realizedPnl from the previous position is not added to the **`marginToRemove()`** variable, but it is removed from the openNotional, leading to loss of yield for the liquidator.

Recommendation

Add the **`=`** symbol to the operation **`marginToRemove += params.rpnl;`**.

3S-NFTPerp-M01

`_getPriceToTick()` reverts if the price is smaller than **`1e10`**, which may happen in **`_search()`** as it assumes the final tick as going entirely to the amm

Id	3S-NFTPerp-M04
Classification	Medium
Severity	Medium
Likelihood	Medium
Category	Bug
Status	Addressed in the AMMRouter

Description

`_search()` calculates the final tick as if the order is completely filled on the amm. This means that if a significant chunk of the liquidity is in the limit orders, if a user has an order bigger than the current amm liquidity, it may return **`1e7`** from amm.getMarkPriceAfterOpen(), which reverts in _getPriceToTick(), as it divides by **`1e10`** and reverts when calculating the tick.

Recommendation

Return **`1e10`** instead of **`1e7`** so that it does not round down to 0 in **`_getPriceToTick()`**.

3S-NFTPerp-M02

`setFundingPeriod()` can be DoSed due to calling `settleFunding()`

Id	3S-NFTPerp-M05
Classification	Medium
Severity	Medium
Likelihood	High
Category	Bug
Status	Addressed in #d75fc8c

Description

setFundingPeriod() calls settleFunding() in the ClearingHouse. It may be the case that someone frontruns **`setFundingPeriod()`** with a call to **`settleFunding()`**, making the **`setFundingPeriod()`** transaction revert.

Recommendation

Make the funding payments pro-rata to time instead of in fixed intervals. This would also solve the fact that **`settleFunding()`** calls can be frontrunned to avoid payments.

3S-NFTPerp-L01

In **`fillPair()`**, when the taker is the maker, the **`reduceOnly`** order mapping is not being deleted

Id	3S-NFTPerp-L05
Classification	Low
Severity	Low
Likelihood	High
Category	Bug
Status	Addressed in #0dd59c3

Description

When filling limit orders, if the trader doing the **`openPosition()`** call is the same as the maker of the limit order, it will delete the limit order.

Recommendation

If the limit order is a **`reduceOnly`** order, it should also delete the reduceOnlyOrders mapping.

3S-NFTPerp-L02

`_getPositionNotional()` crops the size of the position to the available reserves, which may lead to unexpected profits/losses

Id	3S-NFTPerp-L06
Classification	Low
Severity	Medium
Likelihood	Low
Category	Bug
Status	Addressed in #0ac12e1

Description

_getPositionNotional() fills positions to the amm if it can't find limit orders. However, if the remaining size of the position is bigger than the available base reserves in the **`AMM`**, it crops the size of the position (for short positions), decreasing the current notional. It does not seem possible to exploit this situation, as some tests were carried out which led to big losses, but the position could not be closed as there was not enough liquidity, nor could it be liquidated as liquidations use the liquidation price and the not the current value of the position.

Recommendation

Revert if there is not enough liquidity to make sure the value of the position is correctly calculated.

3S-NFTPerp-L03

`getTriggerOrders()` should be paginated or it may revert if enough orders are created

Id	3S-NFTPerp-L07
Classification	Low
Severity	Low
Likelihood	Low
Category	Bug
Status	Acknowledged

Description

There is no limit in the amount of **`triggerOrders`**, which means that the array may grow too large and make transactions revert due to the gas usage exceeding the block gas limit when calling getTriggerOrders().

Recommendation

Paginate the getter so users can fetch information safely.

3S-NFTPerp-L04

Partially removing liquidity may underflow if the price of the pool has changed significantly

Id	3S-NFTPerp-L09
Classification	Low
Severity	Medium
Likelihood	Low
Category	Bug
Status	Addressed in the AMMRouter

Description

removeLiquidity() updates the position quote and base amount based on the removed quote and base amounts. However, if the proportions changed since the position was opened, it may underflow.

Recommendation

Reduce the quote and base amounts of the position pro-rata to the removed shares.

3S-NFTPerp-L05

`removeLiquidity()` in the amm calculates **`marginToRemove`** without updating the margin with the funding payment

Id	3S-NFTPerp-L10
Classification	Low
Severity	Medium
Likelihood	Low
Category	Bug
Status	Addressed in the AMMRouter

Description

removeLiquidity() removes the margin from the position pro-rata to the removed shares, without considering the previous funding payment. The impact should be limited to the case when a significant amount of shares is removed and the **`fundingPayment`** is applied after the margin removal, which may lead to bad debt and revert.

Recommendation

Add the **`fundingPayment`** to the margin and only then calculate **`marginToRemove`**.

3S-NFTPerp-L06

PriceFeed: Two step owner transfers are safer

Id	3S-NFTPerp-L11
Classification	Low
Severity	Low
Likelihood	Low
Category	Suggestion
Status	Acknowledged

Description

When changing contract owner, a two step process is recommended. Using this method, first a pending owner is set, which then has to accept the role in order to become the contract owner. This prevents situations in which a mistake when setting the new owner address leads to permanent loss of contract ownership.

3S-NFTPerp-L07

Total Position Size isn't updated properly

Id	3S-NFTPerp-L13
Classification	Low
Severity	Low
Likelihood	Medium
Category	Bug
Status	Addressed in #a3dc121

Description

The **totalPositionSize** mapping is used to hold the position information for an amm using the **TotalPositionSize** struct, which has the following fields:

- **int256 netPositionSize**
- **uint256 positionSizeLong**
- **uint256 positionSizeShort**

This mapping is updated exclusively through function **_updateTotalPositionSize(...)** of the ClearingHouseBase contract:

```
function _updateTotalPositionSize(IAMM amm, int256 size, Side side) internal
{
    TotalPositionSize memory tps = totalPositionSize[amm];
    tps.netPositionSize += size;
    if (side == Side.LONG) {
        tps.positionSizeLong = size.abs() + tps.positionSizeLong;
    } else {
        tps.positionSizeShort = size.abs() + tps.positionSizeShort;
    }
    totalPositionSize[amm] = tps;
}
```

The issue in this function is that both **tps.positionSizeLong** and **tps.positionSizeShort** are being updated with the absolute value of the size, resulting in values that are always increasing and don't reflect the current size of all long/short positions. For example, if someone opens a long position with a size of 1000 and then closes this position, the **netPositionSize** will go back to zero (as it should) but the **positionSizeLong** will go to 2000 (1000 on creation and 1000 on closure).

Recommendation

Change the **_updateTotalPositionSize()** function to increase/decrease the **tps.positionSizeLong**/**tps.positionSizeShort** variables depending on whether the size is positive or negative:

```
    if (side == Side.LONG) {
        tps.positionSizeLong = size > 0 ?  tps.positionSizeLong +
size.abs(): tps.positionSizeLong - size.abs();
    } else {
        tps.positionSizeShort = size < 0 ? tps.positionSizeShort +
size.abs(): tps.positionSizeShort - size.abs();
    }
```

Tests for the **totalPositionSize** mapping should also be added.

3S-NFTPerp-N01

State changes should always emit events

Id	3S-NFTPerp-N09
Classification	None
Severity	None
Likelihood	High
Category	Bug
Status	Addressed in multiple commits

Description

Some state changes are not emitting events

- https://github.com/nftperp/NFTPerp-V2-Contracts/blob/1c16f2010a851ac23aec73822d55388901bbe5e7/src/ClearingHouse.sol#L636

- https://github.com/nftperp/NFTPerp-V2-Contracts/blob/1c16f2010a851ac23aec73822d55388901bbe5e7/src/ClearingHouse.sol#L645

- https://github.com/nftperp/NFTPerp-V2-Contracts/blob/1c16f2010a851ac23aec73822d55388901bbe5e7/src/ClearingHouse.sol#L654

- https://github.com/nftperp/NFTPerp-V2-Contracts/blob/1c16f2010a851ac23aec73822d55388901bbe5e7/src/AMM.sol#L754

- https://github.com/nftperp/NFTPerp-V2-Contracts/blob/1c16f2010a851ac23aec73822d55388901bbe5e7/src/AMM.sol#L758-L759

- https://github.com/nftperp/NFTPerp-V2-Contracts/blob/1c16f2010a851ac23aec73822d55388901bbe5e7/src/AMM.sol#L765-L766

-
https://github.com/nftperp/NFTPerp-V2-Contracts/blob/1c16f2010a851ac23aec73822d55388901bbe5e7/src/AMM.sol#L791

3S-NFTPerp-N02

Swapping amounts that would send the price below/above the bounds of the amm underflows without a reason

Id	3S-NFTPerp-N10
Classification	None
Severity	None
Likelihood	
Category	Suggestion
Status	Addressed in the AMMRouter

Description

For example, _getBaseValue() calculates the base amount according to the **k** using the virtual reserves; however, some amounts will return a base amount bigger than the real reserves, silently underflowing when updating the reserves in swapOpen().

Recommendation

Add a revert message if the base/quote amount calculated is bigger than the real base/quote reserves.

3S-NFTPerp-N03

PriceFeed: First owner isn't set as valid keeper

Id	3S-NFTPerp-N11
Classification	None
Severity	None
Likelihood	
Category	Suggestion
Status	Acknowledged

Description

When changing contract owner, the **`keeper`** permission of the previous owner is revoked, and this permission is given to the new owner. The issue is that during initialization, the initial owner isn't given this privilege, so they must then call **`setValidKeeper()`** to set themselves (or some other address) as a valid **`keeper`**.

Recommendation

Since the owner should have the power to directly perform **`keeper`** operations, and for consistency with the function **`transferOwnership()`**, it is recommended that this permission is given during ownership initialization, line 45.

3S-NFTPerp-N04

Structs can be packed to save gas

Id	3S-NFTPerp-N12
Classification	None
Severity	None
Likelihood	
Category	Optimization
Status	Acknowledged

Description

In solidity, structs can often be packed into fewer words to save gas on storage loads/stores. In the code, the following structs can be optimized in this way: **`LimitOrder`**, **`TriggerOrder`**, **`TwapInputAsset`**, **`MarketOrder`**.

Recommendation

Reorder the variables in these structs to pack them together.

3S-NFTPerp-N05

Contracts should inherit their interfaces

Id	3S-NFTPerp-N13
Classification	None
Severity	None
Likelihood	Low
Category	Good Code Practices
Status	Acknowledged

Description

To avoid differences between the function declarations in the contracts and their interfaces, it is a good practice for contracts to inherit their interfaces. At the moment the AMM contract doesn't inherit the IAMM interface and this leads to problems of functions existing in the interface but not the contract. For instance, functions **`getInitializationPrice()`**, **`getLiquidationParams()`** and **`getReservesAndTotalLiquidity()`**. In the future a user might try to interact with the protocol using the interface, but the call would certainly revert due to this issue.

3S-NFTPerp-N06

Bug in _reversePosition() might lead to future exploits

Id	3S-NFTPerp-N14
Classification	None
Severity	None
Likelihood	Low
Category	Bug
Status	Addressed in #20cabce

Description

In the **positionManager** library, internal functions to increase, decrease, close or even reverse a position tend to not **revert** and instead rely on a response system to inform the parent function if the call was successful. Such system is used in function **_reversePosition()** which closes and opens a position in the reverse direction, if the traded size is larger that the position's old size. The logic to close and open a new position is presented in the following code segment: (note that some code was omited for simplification)

```
function _reversePosition() internal returns (TradeResponse memory response)
{
  (...)
  closeResponse = _closePosition(...);
  if (closeResponse.makerResult != LimitOrderResult.FILLED) return
closeResponse; // line 1047

  if (notional.div(leverage) != 0) { // if margin of remaining order is
non-zero
      response = _increasePosition(...);
      if (closeResponse.makerResult != LimitOrderResult.FILLED) return
response; // line 1060

      response = TradeResponse({
          position: response.position,
          exchangedQuote: closeResponse.exchangedQuote +
response.exchangedQuote,
          badDebt: closeResponse.badDebt + response.badDebt,
```

```
            exchangedSize: closeResponse.exchangedSize +
response.exchangedSize,
            (...)
            makerResult: response.makerResult
        });
    }
}
```

Here, there is a bug in line 1060, which checks if the call to **`_closePosition()`** (which had already been checked on the line 1047) was filled, instead of the call to **`_increasePosition()`**. This means that lines 1047 and 1060 are checking the same condition, and if the execution reached line 1060, the condition will always be false, leading to unreachable code in line 1060.

At the moment this bug isn't exploitable, since thankfully the function returns the combination of the **`close`** response and the **`increasePosition`** response, and uses the **`makerResult`** of the **`increasePosition`** response (which will be handled correctly by the parent function). There is however, the possibility that a future change exposes this bug, leading to a severe vulnerability.

Recommendation

Change line 1060 to:

```
if (response.makerResult != LimitOrderResult.FILLED) return response;
```

3S-NFTPerp-N07

Using low level pop is not recommended

Id	3S-NFTPerp-N15
Classification	None
Severity	None
Likelihood	
Category	Good Code Practices
Status	Addressed in #af673aa

Description

Using assembly blocks in solidity isn't recommended, unless it presents as obvious advantage, usually an optimization, that can't be performed with standard solidity. Line 214 of the ClearingHouseBase contract implements such an assembly block, popping an element from a memory array by overriding its size.

The issue is that this line, not only makes the code harder to read and audit, but also doesn't seem to present any real advantage compared to simply using the **pop()** function on the array in storage. In regards to gas, this approach actually tends to underperform a much simpler manipulation of the storage array, as proven by the following benchmarks:

```
pragma solidity 0.8.19;
contract test {
    mapping(uint256 trader => mapping(uint256 amm => uint256[] ids))
internal reduceOnlyOrders;
    constructor() {
        reduceOnlyOrders[0][0].push(8);
        reduceOnlyOrders[0][0].push(9);
    }
   // execution cost if length = 2 and id_deleted = 9: 13715 gas
   // execution cost if length = 2 and id_deleted = 8: 16629 gas
   // execution cost if length = 1: 11211 gas
    function test_function() external {
        uint256 id = 8;
        uint256[] memory linkedOrders = reduceOnlyOrders[0][0];
        if (linkedOrders.length == 0) return;
        if (linkedOrders.length == 1) {
            delete reduceOnlyOrders[0][0];
```

```
        } else {
            if (linkedOrders[0] == id) { // if id is first linked order,
swap it
                linkedOrders[0] = linkedOrders[1];
            }
            // pop
            assembly { mstore(linkedOrders, sub(mload(linkedOrders), 1)) }
            reduceOnlyOrders[0][0] = linkedOrders;
        }
    }
}

contract test_optimized {
    mapping(uint256 trader => mapping(uint256 amm => uint256[] ids))
internal reduceOnlyOrders;
    constructor() {
        reduceOnlyOrders[0][0].push(8);
        reduceOnlyOrders[0][0].push(9);
    }
   // execution cost if length = 2 and id_deleted = 9: 13198 gas
   // execution cost if length = 2 and id_deleted = 8: 16784 gas
   // execution cost if length = 1: 10879 gas
    function test_function() external {
        uint256 id = 8;
        uint256 length = reduceOnlyOrders[0][0].length;
        if (length == 0) return;
        if (length == 1) {
            delete reduceOnlyOrders[0][0];
        } else {
            // note: reduceOnlyOrders has a max length of 2
            if (reduceOnlyOrders[0][0][0] == id) { // if id is first linked
order, swap it
                reduceOnlyOrders[0][0][0] = reduceOnlyOrders[0][0][1];
            }
            reduceOnlyOrders[0][0].pop();
        }
    }
}
```

Note: adding a comment stating that the array max size is equal to 2 will also make it easier to understand.

3S-NFTPerp-N08

Errors could include relevant arguments whenever possible, making it easier to debug

Id	3S-NFTPerp-N16
Classification	None
Severity	None
Likelihood	
Category	Good Code Practices
Status	Acknowledged

Description

While during testing it's possible to get the relevant information using other methods (**`console.log`** comes to mind), when the contract is deployed and transactions are live, it's harder to debug without the information being contained in the error.

This could be even more relevant in case of a suspiscious transaction that is being analyzed, where time is very important to prevent damage.

See `ClearingHouse`, for example.

3S-NFTPerp-N09

`mul()` and **`div()`** in **`NFTPMath`** are misleading due to scaling by **`1e18`** underneath

Id	3S-NFTPerp-N17
Classification	None
Severity	None
Likelihood	
Category	Good Code Practices
Status	Acknowledged

Description

Usually **`mul()`** and **`div()`** do multiplication and division, respectively, without scaling by **`1e18`** (commonly used by **`SafeMath`**). However, **`NFTPMath`** `mul()` and `div()` scale the operations by **`1e18`**, misleading code readers.

For example, when looking at **`ClearingHouse:openPosition()`**, it seems that it would round down to 0 in `_assertMaxLeverage()`, but this is not the case as it correctly scales the numbers.

Recommendation

Consider renaming the functions to **`fullMulWad()`** and **`fullDivWad()`**.