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 a 2 person week engagement. The audit was conducted from 29-01-2024 to 02-02-2024.

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
    ├── AMMRouter.sol
    ├── AMMRouterBase.sol
    ├── ClearingHouse.sol
    ├── ClearingHouseBase.sol
    ├── InsuranceFund.sol
    ├── PoolFactory.sol
    ├── PriceFeed.sol
    ├── clearinghouse-libs
    |   ├── Deleverager.sol
    |   ├── OrderCreator.sol
    |   ├── OrderCreatorCompressed.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	8ea05fc0ecf299ca8374b571486846652c8725f8
Language	Solidity
Platform	Arbitrum

Engagement Summary

Timeline	29-01-2024 to 02-02-2024
№ of Auditors	2
Review Time	2 person weeks - after the first 8 person weeks audit.

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	0	0	0
High	0	0	0
Medium	3	3	0
Low	3	3	0
None	9	6	3

Category Breakdown

Suggestion	4
Documentation	0
Bug	6
Optimization	4
Good Code Practices	1

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	Moderate. The codebase was subject to some changes during the course of 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	Satisfactory. No front-running issues were identified.
Monitoring	Satisfactory. Most state changes correctly emit events.
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-M01

Notional difference in AmmRouter:removeLiquidty() may revert in certain cases

Id	3S-NFTPerp-M01
Classification	Medium
Severity	Medium
Likelihood	Medium
Category	Bug
Status	Addressed in #74bc3c3

Description

AmmRouter:removeLiquidity() subtracts the long notional by the short notional if the size is bigger than 0 (contrary if size is smaller than 0), which may revert if the short notional is bigger than the long one.

Recommendation

Use the distance between the notionals.

3S-NFTPerp-M02

It's possible to mint an infinite number of shares without increasing quote or base amounts, due to rounding down

Id	3S-NFTPerp-M02
Classification	Medium
Severity	Medium
Likelihood	Medium
Category	Bug
Status	Addressed in #653fd2c

Description

For very low notional amounts, when adding liquidity, invariantIncrease is always bigger than 0 (not true for the first depositer though), but the corresponding quote and base amounts might be 0. This means that malicious users could loop **`addLiquidity()`** calls, minting a very low amount of shares each time, without ever increasing quote and base. It is most likely not profitable to perform this shares inflation, but some way could be found to exploit it in a way that is profitable. Plugging in some numbers, it was found that it is possible to mint 22 shares with 0 quote and base amount, which if looped enough times could change the pool state significantly.

Recommendation

Add a minimum notional amount when adding liquidity to reduce the attack surface.

3S-NFTPerp-M03

Price deviation as is can be circumvented by making smaller trades in a loop

Id	3S-NFTPerp-M03
Classification	Medium
Severity	Medium
Likelihood	High
Category	Bug
Status	Addressed in #089ec00

Description

The price deviation check will look at the last price snapshots; however, it does not actually enforce that these are from past blocks. Thus, attackers may split a trade in smaller trades and incrementally increase the price.

Recommendation

Store the price of the previous block.

3S-NFTPerp-L01

Call **`_updateMarkPrice()`** whenever **`markPrice`** is changed

Id	3S-NFTPerp-L02
Classification	Low
Severity	Medium
Likelihood	Low
Category	Bug
Status	Addressed in #1423a79

Description

Functions addLiquidity() and removeLiquidity() change mark price but don't call **`_updateMarkPrice()`**. Since these functions are meant to add/remove liquidity, the price isn't expected to fluctuate drastically, but numerical approximations and the use of virtual reserves can lead to some price fluctuation. Calling **`_updateMarkPrice()`** would take a new snapshot of the price ensuring consistency between the **`markPrice`** variable and the snapshots on storage, as well as protect users in case of price fluctuation.

Recommendation

Call **`_updateMarkPrice()`** to update the **`markPrice`** instead of changing this variable directly

3S-NFTPerp-L02

`_match()` always checks the trigger of the first order of a certain tick, instead of checking **`i`** order

Id	3S-NFTPerp-L03
Classification	Low
Severity	Medium
Likelihood	Low
Category	Bug
Status	Addressed in #b471f71

Description

PositionManager:_match() looks for all the orders for their trigger values as some order may have been filled/deleted. However, currently it only checks the first order instead of the **`i`** one, which could cause problems down the line.

3S-NFTPerp-L03

Use a flag to increase or decrease the index of the pool instead of a heuristic

Id	3S-NFTPerp-L04
Classification	Low
Severity	Medium
Likelihood	Low
Category	Bug
Status	Addressed in #68c722c

Description

AmmRouter:_shitPool() increases the index if the price is within 100 of the upper bound of the current pool price. This is error prone as a fill could for example increase the price of a pool, but not be close enough to the upper bound, making the index decrease, entering in a wrong state.

Recommendation

Use the direction flag to increase or decrease the index, as in AmmRouter:_getQuoteValue().

3S-NFTPerp-N01

`PositionManager:_reversePosition()` calculates the notional to reverse but could just use the **`exchangedQuote`**

Id	3S-NFTPerp-N01
Classification	None
Severity	None
Likelihood	
Category	Optimization
Status	Addressed in #669b569

Description

PositionManager:_reversePosition() reduces the notional by the amount required to close the previous position, fetching the value using the getters before. However, it could use **`closeResponse.exchangedQuote()`** instead, which does the same calculations.

3S-NFTPerp-N02

Duplicate size to fill check in _staticFillToAmm()

Id	3S-NFTPerp-N02
Classification	None
Severity	None
Likelihood	
Category	Optimization
Status	Addressed in #1c107ae

Description

_staticFillToAmm() checks twice the size to fill.

Recommendation

Remove the second check as it is never triggered.

3S-NFTPerp-N03

gap is missing the private keyword

Id	3S-NFTPerp-N03
Classification	None
Severity	None
Likelihood	
Category	Suggestion
Status	Addressed in #1b573fa

Description

The abstract contracts use gaps to change storage later if required, but AMMRouterBase is missing the **`private`** keyword.

3S-NFTPerp-N04

Index underflow is not protected against, although it has no impact as the pool with index type(uint256).max should not be registered

Id	3S-NFTPerp-N04
Classification	None
Severity	None
Likelihood	
Category	Suggestion
Status	Acknowledged

Description

The following places do an unchecked increment/decrement, but the index could underflow and become **`type(uint256).max`**. This has no consequences with the code as is, but it's better to revert for underflows.

https://github.com/nftperp/NFTPerp-V2-Contracts/blob/8ea05fc0ecf299ca8374b571486846652c8725f8/src/AMMRouter.sol#L667

https://github.com/nftperp/NFTPerp-V2-Contracts/blob/8ea05fc0ecf299ca8374b571486846652c8725f8/src/AMMRouter.sol#L1087-L1092

https://github.com/nftperp/NFTPerp-V2-Contracts/blob/8ea05fc0ecf299ca8374b571486846652c8725f8/src/ClearingHouseBase.sol#L477

3S-NFTPerp-N05

In AmmRouter:liquidateMaker(), the pools array can safely be deleted from storage

Id	3S-NFTPerp-N05
Classification	None
Severity	None
Likelihood	
Category	Optimization
Status	Addressed in #0ea5723

Description

`AmmRouter:liquidateMaker()` calls _unregisterPoolsFromMaker() with all the pools, which means that it could simply delete the pools from storage.

3S-NFTPerp-N06

trimDuplicatePools() is very expensive, having O(n^2) complexity and could be simplified

Id	3S-NFTPerp-N06
Classification	None
Severity	None
Likelihood	
Category	Optimization
Status	Addressed in #8c1e930

Description

ammRouter:trimDuplicatePools() eliminates duplicate pools in the input by comparing every element of the array for duplicates, which is very expensive.

Note: the function is actually useless as it is impossible to add liquidity in duplicate pools. It would revert with error "SANDWICH".

Recommendation

Force the user to input pool indexes ordered and if there are duplicates, revert. Here is an example of safe using this technique.

3S-NFTPerp-N07

Add a 0 address check on the pool when adding liquidity for greater verbosity

Id	3S-NFTPerp-N07
Classification	None
Severity	None
Likelihood	
Category	Good Code Practices
Status	Addressed in #51fb761

Description

AmmRouter:addLiquidity() does not have an explicit check regarding the existence of the pool with the given id, making it only revert when calling **`poolsAdded[i].addLiquidity()`** due to internal EVM checks.

Recomendation

Place an explicit check for this situation.

3S-NFTPerp-N08

setPools() could use more validation

Id	3S-NFTPerp-N08
Classification	None
Severity	None
Likelihood	
Category	Suggestion
Status	Acknowledged

Description

`AmmRouter:setPools()` is an admin function but could still benefit from input validaiton to prevent mistakes. For example, specifying a pool id that already exists will overwrite the current amm, which could be dangerous. Additionally, it should be checked that the bounds of the pools increase with the **`index`**, which is currently not enforced but the logic depends on it.

3S-NFTPerp-N09

Misleading error name

Id	3S-NFTPerp-L01
Classification	None
Severity	None
Likelihood	
Category	Suggestion
Status	Acknowledged

Function closePosition() of the **`ClearingHouse`** contract checks if the size sent as an argument to this function is smaller than the size of the position to close. If the size to close is greater than the position size, the execution reverts with error **`INSUFFICIENT_SIZE`**. This error name is misleading and could lead a user into thinking they called the function with insufficient size, and call the same function again with a greater size, which would lead to another revert. For clarification, take a look at a simplified code segment illustrating this issue

```
function closePosition(IRouter amm,  uint256 size, uint256 quoteLimit)
external {
    int256 oldSize = _assertOpenPosition(amm, msg.sender);
    uint256 oldSizeAbs = oldSize.abs();
    if (size > oldSizeAbs) revert INSUFFICIENT_SIZE();
    (...)
}
```