Three Sigma

Code Audit

Vertex CEX speed, DeFi security combined.

Disclaimer

Code Audit

Vertex CEX speed, DeFi security combined.

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

Vertex CEX speed, DeFi security combined.

Table of Contents

Summary

Code Audit

Vertex CEX speed, DeFi security combined.

Summary

Three Sigma audited Vertex in a 14 person week engagement. The audit was conducted from 01-04-2024 to 10-05-2024.

Protocol Description

Vertex is a decentralized exchange (DEX) offering a comprehensive product suite, including spot and perpetual markets, lending, borrowing, and yield generation, all powered by a hybrid orderbook and AMM model. Built for performance, Vertex combines the speed and efficiency of centralized exchanges (CEXs) with the security and transparency of decentralized finance (DeFi). With deep liquidity, low fees, and an advanced feature set, Vertex is designed to deliver a seamless trading experience for both retail and institutional users.

Scope

Code Audit

Vertex CEX speed, DeFi security combined.

Scope

Filepath	nSLOC
contracts/ArbAirdrop.sol	84
contracts/BaseEngine.sol	284
contracts/Clearinghouse.sol	415
contracts/ClearinghouseLiq.sol	628
contracts/ClearinghouseStorage.sol	102
contracts/common/Constants.sol	18
contracts/common/Errors.sol	51
contracts/Endpoint.sol	734
contracts/EndpointGated.sol	26
contracts/libraries/MathSD21x18.sol	89
contracts/libraries/RiskHelper.sol	107
contracts/OffchainExchange.sol	710
contracts/PerpEngine.sol	241
contracts/PerpEngineLp.sol	156
contracts/PerpEngineState.sol	191
contracts/ProxyManager.sol	172
contracts/SpotEngine.sol	286
contracts/SpotEngineLP.sol	175
contracts/SpotEngineState.sol	335
contracts/Verifier.sol	173
contracts/Version.sol	8
Sum	**4985**

Assumptions

OpenZeppelin is considered secure.

Methodology

Code Audit

Vertex CEX speed, DeFi security combined.

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

Vertex CEX speed, DeFi security combined.

Project Dashboard

Application Summary

Name	Vertex
Commit	50601071fb44010524ce4d667c5c881af1b2533e
Language	Solidity
Platform	Arbitrum

Engagement Summary

Timeline	01-04-2024 to 10-05-2024
№ of Auditors	2
Review Time	14 person weeks

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	1	1	0
High	3	3	0
Medium	1	0	1
Low	16	4	12

None	3	0	3

Category Breakdown

Suggestion	9
Documentation	0
Bug	14
Optimization	0
Good Code Practices	1

Code Maturity Evaluation

Code Audit

Vertex CEX speed, DeFi security combined.

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 as well as takes the correct measures in rounding the results of arithmetic operations.
Centralization	Moderate. The sequencer and protocol parameters are admin-controlled, but users can still manually submit transactions.
Code Stability	Satisfactory. No new functionality was being added throughout the audit.
Upgradeability	Satisfactory. The protocol is upgradeable.
Function Composition	Satisfactory. Functionality was well split into functions.
Front-Running	Satisfactory. No significant frontrunning opportunities were found.
Monitoring	Moderate. Some state changes were not emitting events.
Specification	Satisfactory. The code matched the design specifications.
Testing and Verification	Satisfactory. Unit and integration tests were present.

Findings

Code Audit

Vertex CEX speed, DeFi security combined.

Findings

3S-Vertex-C01

`ClearingHouseLiq::_assertLiquidationAmount()` may increase basis points due to negative `quoteBalance.amount + insurance`

Id	3S-Vertex-C01
Classification	Critical
Severity	Critical
Likelihood	Medium
Category	Bug
Status	Addressed in #38bbe76.

Description

If the position is a spread, it may be liquidated as one up to the minimum absolute value of the spot and perp positions. If the spot is short, the `liquidatee + insurace` need to have enough `quote` to buy back the spot position. This check is performed in `ClearingHouseLiq::_assertLiquidationAmount()`:

```
...
basisAmount = MathHelper.max(
    -((quoteBalance.amount + insurance).div(
        liquidationPrice
    ) + 1),
    basisAmount
);
...
```

Note that `quoteBalance.amount + insurance` may be negative, which would mean that `abs(basisAmount)` could be increased, leading to incorrect states.

Recommendation

Add `if (quoteBalance.amount + insurance <= 0) basisAmount = 0` or similar.

3S-Vertex-H01

`state.cumulativeDepositsMultiplierX18` may become **`0`** or **`negative`**, leading to loss of funds

Id	3S-Vertex-H01
Classification	High
Severity	Critical
Likelihood	Low
Category	Bug
Status	Addressed in #38bbe76.

Description

`state.cumulativeDepositsMultiplierX18` may go below **`0`** if an account is socialized or in the worst case, **`0`**, which will have greater impact. The point of socializing a spot account is eliminating the borrowed amount from the **`short`** account and transferring it to the other depositors. The problem is that the multiplier may become **`0`** or **`negative`** which is never handled. If the multiplier goes to **`0`**, every deposit will revert and the user will lose its funds. If it becomes negative, every deposit will be treated as a short instead and the user may be liquidated (deposits have no health check).

Recommendation

Don't let the multiplier become **`0`** or lower by reverting in this case.

3S-Vertex-H02

`OffchainExchange::swapAmm()` does not validate that `txn.priceX18 > 0`, allowing donation attacks

Id	3S-Vertex-H02
Classification	High
Severity	Critical
Likelihood	Low
Category	Bug
Status	Addressed in #38bbe76.

Description

`OffchainExchange::swapAmm()` calculates the quote amount as `-txn.amount.mul(txn.priceX18)`. Thus, if the price is negative, base and quote amount will have the same sign. If it's negative, the transaction will revert as it would lead to a smaller `k` in the `amm`. However, if it's positive, it will provide both base and quote to the `amm` and the user will take a big loss.

Using a `txn.price < 0`, it can be used to perform donation attacks, which could be dangerous, as it maintains the `base/quote` ratio but inflates lp value. An example of an attack is the following:

Price = 10, attacker mints 11 lp using 1 base and 10 quote.

Attacker calls `OffchainExchange::swapAmm()` with `txn.amount = 1e21` and `txn.price = -10`. This changes the amm state to 11 lp, approx 1e21 base and 1e22 quote. Some user adds lp with 1e20 base and 1e21 quote (note that the price is maintained)

`toMint = 1e20/1e21*11`, which rounds down to 0, stealing the funds.

Additionally, it is recommended to enforce that swaps follow the amm curve to prevent price manipulations.

Recommendation

Ensure that the price is positive, as a negative price would be a user mistake or a donation attack.

3S-Vertex-H03

`ClearingHouseLiq::_finalizeSubaccount()` does not check if the `subaccount` has `lps`

Id	3S-Vertex-H03
Classification	High
Severity	High
Likelihood	Medium
Category	Bug
Status	Addressed in #38bbe76.

Description

`ClearingHouseLiq::liquidateSubaccountImpl()` can finalize an account, repaying the bad debt from insurance or socializing the account, depending on the amount of insurance available. It checks that the account has no positive balances, otherwise it may receive insurance or be socialized in some assets, increase its health and then retrieve the positive balances.

However, it does not check if the **`subaccount`** has **`lps`**, which may be used to maliciously steal funds from insurance or via socialization of the debt. This can be exploited as the healh of the **`lp`** position should be lower than the real worth of the pro-rata base and quote assets, due to how the geometric mean works, which means that even if **`ClearingHouseLiq::liquidateSubaccountImpl()`** requires the account to be unhealthy at the end, the attacker will still profit later by burning the lp for the underlying amounts.

Recommendation

In **`ClearingHouseLiq::_finalizeSubaccount()`**, revert if the **`subaccount`** has lp balances.

3S-Vertex-M01

Missing **`disableInitializers()`** call in the constructor

Id	3S-Vertex-M01
Classification	Medium
Severity	Medium
Likelihood	Medium
Category	Suggestion
Status	Acknowledged

Description

When using **`Initializable.sol`**, it's a good practice calling **`disableInitializers()`** in the constructor, such that the implementation itself can't be initialized. The call is missing in the Clearinghouse.sol, Endpoint.sol, OffchainExchange.sol, PerpEngine.sol, SpotEngine.sol, ClearinghouseLiq.sol and Verifier.sol (commented out).

ClearingHouse may be selfdestructed via the delegateCall().

Recommendation

Use:

```
constructor() {
     _disableInitializers();
}
```

3S-Vertex-L01

`Verifier::checkQuorum()` returns **`false`** with more than 3 signers

Id	3S-Vertex-L01
Classification	Low
Severity	Low
Likelihood	High
Category	Bug
Status	Addressed in #38bbe76.

Description

The signerBitmask is hardcoded to 7 when calling requireValidSignature(), meaning that nSigned would be incremented at most 3 times in checkQuorum() as **`7 = 00000111`**. Consequently, if there are more than 3 signers added in the **`Verifier`** contract, **`Verifier::checkQuorum()`** will always return **`false`** in the following scenarios:

- **`nSigner >= 4`** and **`nSigned <= 2`**
- **`nSigner >= 6`** and **`nSigned <= 3`**

Recommendation

Make **`signerBitmask`** dynamic.

3S-Vertex-L02

Downcasting Leads to Silent Overflow

Id	3S-Vertex-L02
Classification	Low
Severity	Medium
Likelihood	Low
Category	Bug
Status	Acknowledged

Description

Variables of type **`int128`** are being upcasted to **`int256`** and then downcasted again in the **`burnLp`** function in PerpEngineLp.sol and SpotEngineLP.sol, leading to silent overflow.

Recommendation

Consider using a solution like OpenZeppelin's SafeCast Library.

3S-Vertex-L03

`isHealthy()` Function Always Returns True

Id	3S-Vertex-L03
Classification	Low
Severity	Low
Likelihood	High
Category	Bug
Status	Acknowledged

Description

The isHealthy function in **`OffchainExchange.sol`** consistently returns **`true`** regardless of its parameter.

Recommendation

While the function is marked as **`virtual`**, there are currently no contracts inheriting and overriding **`OffchainExchange.sol`**. Consider adding meaningful implementation.

3S-Vertex-L04

Precision loss due to engines not using **`.muldiv()`** when calculating **`lp`** ratios

Id	3S-Vertex-L04
Classification	Low
Severity	Low
Likelihood	High
Category	Bug
Status	Acknowledged

Description

For example, in **`SpotEngineState::_getInLpBalance()`**, the ratio is calculated first, likely incurring in precision loss as **`lpState.supply`** is expected to be bigger than **`1e18`** (used in **`.div()`**). This means that **`baseAmount`** and **`quoteAmount`** will be smaller due to the precision loss.

Recommendation

Everytime a division and multiplication are to be applied and division loss could occur (when the denominator is bigger than **`1e18`**), mulDiv() should be used.

https://github.com/vertex-protocol/vertex-contracts-3sigma-audit/blob/main/contracts/SpotEngineLP.sol#L38

https://github.com/vertex-protocol/vertex-contracts-3sigma-audit/blob/main/contracts/PerpEngineState.sol#L115

https://github.com/vertex-protocol/vertex-contracts-3sigma-audit/blob/main/contracts/PerpEngine.sol#L174

https://github.com/vertex-protocol/vertex-contracts-3sigma-audit/blob/main/contracts/PerpEngine.sol#L177

3S-Vertex-L05

Unused argument in **`Endpoint::registerTransferableWallet()`**

Id	3S-Vertex-L05
Classification	Low
Severity	Low
Likelihood	High
Category	Bug
Status	Addressed in #38bbe76.

Description

`Endpoint::registerTransferableWallet()` sets **`transferableWallets[wallet]`** to **`true`**, but a **`_transferable`** argument can be provided.

Recommendation

Delete the argument or replace **`true`** by **`_transferable`**.

3S-Vertex-L06

Risk parameters should include additional checks to prevent mistakes

Id	3S-Vertex-L06
Classification	Low
Severity	Low
Likelihood	Low
Category	Suggestion
Status	Addressed in #38bbe76.

Description

Risk parameters should be lower, higher or equal to **`1e18`**, depending if short or long. For example, the long risk values should be smaller or equal to **`1e18`**, or some of the functionality may get corrupted. In **`SpotEngine::decomposeLps()`**, rewards are calculated as **`amountQuote * (1e18 - short/longMaintenanceWeight)`**. **`amountQuote`** is expected to be a positive value, thus the selected weight will be **`long`**. Thus, if **`longWeightMaintenanceX18 > 1e18`**, rewards will be negative and the **`liquidatee`** earns these rewards.

Recommendation

Place explicit checks when setting the risk parameters in **`BaseEngine::_addProductForId()`**.

3S-Vertex-L07

`PerpEngineLp::burnLp()` and `SpotEngineLp::burnLp()` are missing slippage checks

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

Description

`PerpEngineLp::burnLp()` and `SpotEngineLp::burnLp()` have no slippage control parameters, which means that a user could get `base` and `quote` at a worse price/quantity than expected.

The severity in Vertex is lower as transactions have to go through a delay or the sequencer, which makes it much harder to do MEV. In any case, the issue persists.

Recommendation

Include minimum quote and base parameters in the `burnLp()` functions.

3S-Vertex-L08

`ClearingHouse::addEngine()` initializes **`productEngine`**, so it may be frontrunned and someone initializes **`productEngine`** instead

Id	3S-Vertex-L08
Classification	Low
Severity	Low
Likelihood	Low
Category	Suggestion
Status	Acknowledged

Description

`ClearingHouse::addEngine()` initializes **`productEngine`**, which means it is uninitialized prior to the call. Thus, someone may spot the uninitialized **`productEngine`** proxy and initialize it themselves, making the **`addEngine()`** call revert.

Recommendation

Send the **`productEngine`** creation and **`ClearingHouse::addEngine()`** atomically, via a multicall or custom contract deployer.

3S-Vertex-L09

Any user may enforce the referral code of other new users

Id	3S-Vertex-L09
Classification	Low
Severity	Low
Likelihood	Low
Category	Bug
Status	Addressed in #38bbe76.

Description

In **`Endpoint::depositCollateralWithReferral()`**, the referral is set to **`DEFAULT_REFERRAL_CODE`** if it is a remote deposit (**`sender != address(bytes20(subaccount))`**). Thus, any user may maliciously set other subaccount referral code to default. It's also possible to do this by submitting a slow mode transaction of type **`BurnLpAndTransfer`**.

Recommendation

The specific fix depends on how the referral code is supposed to be handled.

3S-Vertex-L10

`txType == TransactionType.ExecuteSlowMode` may cause batch of transactions to fail due to `_slowModeConfig.txUpTo < _slowModeConfig.txCount`

Id	3S-Vertex-L10
Classification	Low
Severity	Medium
Likelihood	Low
Category	Bug
Status	Acknowledged

Description

`txType == TransactionType.ExecuteSlowMode` allows the sequencer to execute slow mode transactions. However, these transactions can also be executed permissionlessly. `Endpoint::_executeSlowModeTransaction` reverts if `_slowModeConfig.txUpTo < _slowModeConfig.txCount`, which means that if before executing `Endpoint::submitTransactionsChecked()`, someone executes the slow mode transactions up to the latest, the whole batch will revert.

Recommendation

This is unlikely to happen as there is a significant delay in slow mode transactions, which is ignored by the sequencer. Nontheless, an early return can be performed instead of reverting in `Endpoint::_executeSlowModeTransaction`.

3S-Vertex-L11

User deposits are exposed to protocol risk before being able to do anything

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

Description

`depositCollateral()` and related functions transfer funds from the user, but only after the delay or the sequencer submission is the balance of the user updated. This poses a risk for the user that may lose the funds without having the opportunity to do something about them. A simple example would be having a single user in the protocol who deposited, accuring deposit multiplier. If this user withdrew, it would get some of the pending deposit collateral.

Recommendation

`Endpoint::handleDepositTransfer()` could move the funds to a temporary address and only when the deposit is submitted are the funds transferred to the clearing house.

3S-Vertex-L12

Ratios in **`PerpEngineLp`** or **`SpotEngineLp`** may be manipulated when the liquidity is low

Id	3S-Vertex-L12
Classification	Low
Severity	Medium
Likelihood	Low
Category	Suggestion
Status	Acknowledged

Description

PerpEngineLp::mintLp() assigns quote according to

```
int128 amountQuote = (lpState.base == 0)
    ? amountBase.mul(_risk(productId).priceX18)
    : amountBase.mul(lpState.quote.div(lpState.base));
```

with the requirement that **`amountBase % sizeIncrement == 0`**.

In PerpEngineLp::burnLp(), the amounts received are

```
amountBase = MathHelper.floor(
    int128((int256(amountLp) * lpState.base) / lpState.supply),
    sizeIncrement
);
amountQuote = int128(
    (int256(amountLp) * lpState.quote) / lpState.supply
);
```

Thus, with low enough liquidity, the price of the perp engine may be lowered close to 0 by exploiting **`Math.floor()`** or similar strategies and increased up to **`1/sizeIncrement`**. See the poc for details.

The same can be said for the spot engine, with the upper limit of the price being 1.

Note: due to how the health calculation works, changing the prices always leads to less health factor and pnl so it's likely this can not be used to exploit for profit, but users may still be griefed.

Recommendation

Do an initial lp provision to prevent price manipulations.

3S-Vertex-L13

Reevaluate the Need for `renounceOwnership()`

Id	3S-Vertex-L13
Classification	Low
Severity	Low
Likelihood	Low
Category	Suggestion
Status	Acknowledged

Description

Consider overriding OpenZeppelin's Ownable `renounceOwnership()` function if the project's roadmap doesn't plan to relinquish total ownership control.

3S-Vertex-L14

Missing pagination for some functions that iterate

Id	3S-Vertex-L14
Classification	Low
Severity	Low
Likelihood	Low
Category	Bug
Status	Acknowledged

Description

ArbAirdrop::getClaimed() may revert after enough weeks have passed due to OOG.

BaseEngine::getHealthContribution() may revert if enough **`productIds`** are set.

Recommendation

Add initial and final weeks arguments.

3S-Vertex-L15

`Ownable2Step` is preferred over **`Ownable`**

Id	3S-Vertex-L15
Classification	Low
Severity	Low
Likelihood	Low
Category	Suggestion
Status	Acknowledged

Description

Ownable2Step places some important checks, such as a 2 step ownership transfer procedure and should be preferred over Ownable.

Recommendation

Replace **`Ownable2Step`** by **`Ownable`** whenever possible.

3S-Vertex-L16

Missing events

Id	3S-Vertex-L16
Classification	Low
Severity	Low
Likelihood	High
Category	Suggestion
Status	Acknowledged

Description

Some state changes are missing events:

- ArbAidrop, all functions.

Recommendation

Emit events for all relevant state changes.

3S-Vertex-N01

`IERC20Base` Noncompliant With ERC20 Standard

Id	3S-Vertex-N01
Classification	None
Category	Bug
Status	Acknowledged

Description

The interface declares increaseAllowance and decreaseAllowance functions, which aren't part of the official ERC20 standard. Calls to such functions may revert with some tokens. However, these functions are part of the OpenZeppelin's ERC20 implementation designed to mitigate well-known issues around setting allowances. It's important to note that these functions do not entirely fix these issues. An example is **`USDT`** and its "approval race protection," which causes other issues.

Additionally, the **`decimals()`** function is optional, and contracts mustn't rely on it.

Recommendation

It's advisable to use OpenZeppelin's SafeERC20 lib, not only for dealing with allowances but also with transfers, without reinventing the wheel, as seen in **`ERC20Helper.sol`**.

3S-Vertex-N02

Use of `abi.encodePacked()` with Dynamic Types

Id	3S-Vertex-N02
Classification	None
Category	Suggestion
Status	Acknowledged

Description

Using `abi.encodePacked()` with dynamic types, as seen in `Verifier.sol` on L160 and in `Endpoint.sol` on L749, isn't advisable when feeding the outcome into a hashing function like `keccak256()`, due to hash collisions.

Recommendation

Use `abi.encode()` instead.

3S-Vertex-N03

`RiskHelper::isoGroup()` returns 0, regardless of the passed `subaccount`

Id	3S-Vertex-N03
Classification	None
Category	Good Code Practices
Status	Acknowledged

Description

`RiskHelper::isoGroup()` is called in various functions across the code such as RiskHelper::canTrade(), SpotEngine::socializeSubaccount(), PerpEngine::socializeSubaccount(), ClearinghouseLiq::_assertLiquidationAmount(), ClearinghouseLiq::_assertCanLiquidateLiability(), ClearinghouseLiq::_finalizeSubaccount(), and ClearinghouseLiq_settlePositivePerpPnl(). However, it always returns 0 regardless of its parameter `subaccount`.

Recommendation

Implement the functionality (currently commented out) or fully remove this function.