Vertex Protocol

Audit

Presented by:

OtterSec contact@osec.io

Robert Chen r@osec.io

Ajay Kunapareddy d1r3wolf@osec.io

Shiva Shankar sh1v@osec.io

Contents

Appendices

01 | Executive Summary

Overview

Vertex Protocol engaged OtterSec to perform an assessment of the `vertex-evm` program. This assessment was conducted between November 21st and December 20th, 2022.

Critical vulnerabilities were communicated to the team prior to the delivery of the report to speed up remediation. After delivering our audit report, we worked closely with the team to streamline patches and confirm remediation. We delivered final confirmation of the patches January 12th, 2023.

Key Findings

Over the course of this audit engagement, we produced 42 findings total.

In particular, we found multiple critical issues which could lead to the loss of funds. For example, we noted issues with access control (OS-VTX-ADV-01), calculation errors (OS-VTX-ADV-00, OS-VTX-ADV-02, OS-VTX-ADV-03), as well as various issues with fees, liquidations, and denial of service concerns.

We also made numerous recommendations, such as refactoring code redundancies in several parts of the code (OS-VTX-SUG-00), as well as various minor rounding issues (OS-VTX-SUG-07).

Overall, we commend the Vertex team for being responsive and knowledgeable throughout the audit.

02 | Scope

The source code was delivered to us in a git repository at github.com/vertex-protocol/vertex-evm. This audit was performed against commit `e634e80`.

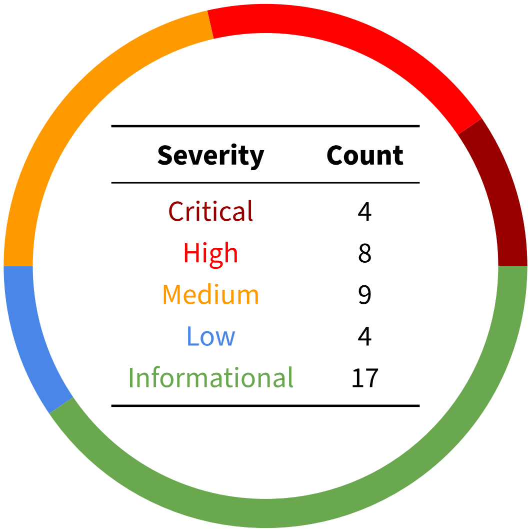
A brief description of the programs is as follows.

Name	Description
vertex-evm	VERTically integrated EXchange (VERT-EX) built on Arbitrum with an orderbook and advanced risk engine to support cross-margin spot and derivatives trading, as well as a money market to facilitate leverage and yield products. In addition to the on-chain components, Vertex leverages an off-chain sequencer to properly simulate orders and efficiently manage risk.

03 | Findings

Overall, we report 42 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings don't have an immediate impact but will help mitigate future vulnerabilities.



Severity	Count
Critical	4
High	8
Medium	9
Low	4
Informational	17

04 | Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-VTX-ADV-00	Critical	Resolved	`AmmEquilibrium` uses `toInt` instead of `fromInt`, which results in the division of the outcome by 1e18..
OS-VTX-ADV-01	Critical	Resolved	The spot and perp engines allow anyone to call `updateStates`, which is a sensitive operation.
OS-VTX-ADV-02	Critical	Resolved	The function `getSettlementState` calculates position profit and loss by considering the entire pool's base and quote assets.
OS-VTX-ADV-03	Critical	Resolved	The method `MathHelper.swap` applies `toInt`, which converts X18 format into an integer, twice on `quoteSwappedX18`.
OS-VTX-ADV-04	High	Resolved	The `socializeSubAccount` function in `PerpEngine` does not save or store the updated product state in the storage.
OS-VTX-ADV-05	High	Resolved	The incorrect calculation of fee amounts in `OffchainBook` results in an inconsistency in total funds.
OS-VTX-ADV-06	High	Resolved	The rounding for `baseSwapped` in `MathHelper` library's swap function fails when the amount is negative.
OS-VTX-ADV-07	High	Resolved	The `dumpFees` function in `OffchainBook` is unreachable, which results in the locking of the collected fees.

OS-VTX-ADV-08	High	Resolved	The funds taken during the submission of `DepositCollateral` transactions are not being returned when transactions fail.
OS-VTX-ADV-09	High	Resolved	`SocializeSubaccount` covers more loss than `insuranceCoverX18`, which could potentially result in a loss of funds.
OS-VTX-ADV-10	High	Resolved	In `SpotEngineStates`’ `updateStates` function, the calculation for `borrowRate` is behaving improperly in certain cases.
OS-VTX-ADV-11	High	Resolved	The `updateStates` function in `SpotEngineState` contract uses the `quoteState` variable instead of the `state` variable.
OS-VTX-ADV-12	Medium	Resolved	Overflow and underflow checks for arithmetic functions in `MathHelper Library` fail for negative numbers.
OS-VTX-ADV-13	Medium	Resolved	Allowing any slow mode transaction with sender value in the transaction struct as an endpoint.
OS-VTX-ADV-14	Medium	Resolved	The `DumpFees` function in `OffchainBook` fails to transfer funds in the case of `PerpEngine`.
OS-VTX-ADV-15	Medium	Resolved	An incorrect cumulative value was passed to `updateBalance` for the pool account.
OS-VTX-ADV-16	Medium	Resolved	The `burnLp` and `mintLp` functions lack checks to ensure that the amounts are positive.
OS-VTX-ADV-17	Medium	Resolved	The `SpotEngineState`’s `updateBalance` function used an incorrect `cumulativeMultiplier` while removing the normalized amount.
OS-VTX-ADV-18	Medium	Resolved	If the pool supply is zero, the base price is derived from the user’s input values.

OS-VTX-ADV-19	Medium	Resolved	In `SpotEngine`'s `socializeSubaccount` function, the amount is added to the total borrow instead of being removed.
OS-VTX-ADV-20	Medium	Resolved	The updating condition for `canSocialize` appears to be incorrect in `Clearinghouse` contract's `getLiquidationStatus` function.
OS-VTX-ADV-21	Low	Resolved	No checks are set to ensure that the `newHealthGroup` is greater than one by `maxHealthGroup`.
OS-VTX-ADV-22	Low	Resolved	In the `swap` function, if the value of `quoteSwappedX18` is positive, it should be rounded upwards.
OS-VTX-ADV-23	Low	Resolved	Additional checks should be added to the price and time to prevent unexpected errors from affecting the protocol.
OS-VTX-ADV-24	Low	Resolved	The `addEngine` function in `ClearingHouse` does not check whether the given engine address is a zero address.

OS-VTX-ADV-00 [crit] | Improper Unit Conversion In AmmEquilibrium

Description

In `ammEquilibrium`, `toInt` was used instead of `fromInt`.

contracts/libraries/MathHelper.sol SOLIDITY

```solidity
function ammEquilibrium(
    int256 baseX18,
    int256 quoteX18,
    int256 priceX18
) internal pure returns (int256, int256) {
    if (baseX18 == 0 && quoteX18 == 0) {
        return (0, 0);
    }
    int256 k = baseX18.toInt() * quoteX18.toInt();
    // base * price * base == k
    // base = sqrt(k / price);
    int256 base = (MathHelper.sqrt(k) * 1e9) / MathHelper.sqrt(priceX18);
    // TODO: this can cause a divide by zero
    int256 quote = k / base;
    return (base.fromInt(), quote.toInt());
}
```

In `PRBMath`:

1. `fromInt` - adds 1e18 precision to integers (multiplication by 1e18).
2. `toInt` - removes 1e18 decimal units to integers (division by 1e18).

Instead of multiplying, it is dividing the `quote` value with 1e18, causing the value to greatly reduce. Since the values returned by this function are sensitive, an attacker may abuse this to manipulate the protocol in order to receive a profit.

Remediation

Change the function call to `fromInt`.

Patch

Function call changed to `fromInt`. Fixed in #116

OS-VTX-ADV-01 [crit] | Unauthorized Access To UpdateStates

Description

In both spot and perp engines, the `updateStates` functions update their respective states based on changes in the market and changes in time (dt). These functions should only be called from `EndPoint`, as it tracks the `lastUpdated` time of the engines and provides the correct dt value.

However, these functions are not restricted by the `onlyEndpoint` modifier, which means that any user can call them with any dt value. This creates a vulnerability that attackers can exploit to manipulate profits by using a manipulated dt value.

Remediation

Add the `onlyEndpoint` modifier to `updateStates` of both spot and perp engines.

Patch

`onlyEndpoint` modifier was added to the `updateStates` functions. Fixed in #116

OS-VTX-ADV-02 [crit] | Improper Settlement Calculation

Description

In `PerpEngine`, `getSettlementState` is used to calculate the amount that users can settle at that time by calculating the total profit/loss that they made in the perp market.

contracts/PerpEngine.sol SOLIDITY

```solidity
(int256 ammBaseX18, int256 ammQuoteX18) = MathHelper.ammEquilibrium(
    lpState.base.fromInt(),
    lpState.quote.fromInt(),
    priceX18
);
int256 positionPnlX18 = priceX18.mul(balance.amountX18 + ammBaseX18) +
    balance.vQuoteBalanceX18 +
    ammQuoteX18;
```

For profit and loss calculations, the entire pool's base and quote balance is considered, whereas the user's portion (`lpBalance`/supply) of the amount in it only needs to be considered. Every user will receive the entire liquidity pool's base and quote value in their profit, leading to a drain of funds and inconsistencies.

Remediation

Consider `amountLp/supply` ratio of pool liquidity for calculating the user's PNL.

Patch

Position PNL calculation is changed to consider only `amountLp/supply` ratio of `ammBase` and `ammQuote` amounts. Fixed in #130

OS-VTX-ADV-03 [crit] | Improper Conversion Of Units In Swap

Description

In MathHelper Library, swap returns to base and quote amounts in which the quote was converted into an integer twice. For example, toInt was applied on quoteSwappedX18 twice in the function, dividing the actual integer's value with 1e18 again.

contracts/libraries/MathHelper.sol SOLIDITY

```solidity
int256 quoteSwappedX18 = (k / (base + baseSwapped) - quote).fromInt();
if (amountSwap > 0) {
    quoteSwappedX18 = quoteSwappedX18.mul(keepRateX18).toInt();
} else {
    quoteSwappedX18 = quoteSwappedX18.div(keepRateX18).toInt();
}
return (baseSwapped.fromInt(), quoteSwappedX18.toInt().toInt());
```

An attacker may abuse this to manipulate the quoted value by which they steal the funds using swapAMM without paying the quote balance.

Proof of Concept

The attacker swaps the quote to base, for which the quote balance a user would have to pay becomes zero if the value is less than 1e18, This is because the amount was divided by 1e18 again. Therefore, no quote balance is deduced from the attacker's account.
An attacker may receive the base amounts for free and withdraw them, throwing the entire protocol into an unresolved state.

Remediation

Remove one of the two toInt() functions over quoteSwappedX18.

Patch

toInt was converted to fromInt at the return. Fixed in #116

OS-VTX-ADV-04 [high] | Updated States Not Stored In SocializeSubAccount

Description

In `PerpEngine`'s `socializeSubAccount` function, the amount is socialized by applying it to the state's `cumulativeFunding`.

contracts/PerpEngine.sol SOLIDITY

```solidity
for (uint256 i = 0; i < productIds.length; ++i) {
    uint32 productId = productIds[i];
    (State memory state, Balance memory balance) = getStateAndBalance(
        productId,
        subaccountId
    );
    if (balance.vQuoteBalanceX18 < 0) {
        ...
        if (balance.vQuoteBalanceX18 < 0) {
            ...
            state.cumulativeFundingLongX18 += fundingPerShareX18;
            state.cumulativeFundingShortX18 -= fundingPerShareX18;
            balance.vQuoteBalanceX18 = 0;
            emit SocializeProduct(productId, -balance.vQuoteBalanceX18);
        }
        balances[productId][subaccountId] = balance;
    }
}
```

However, the updated state was not getting stored back into the storage, fading away the changes. On the other hand, a negative amount from the `userAccount` was cleared, leading to an inconsistency in total funds.

Remediation

Update the state at the end of the loop iteration.

Patch

`state` was stored to the `states[productId]`. Fixed in #116

OS-VTX-ADV-05 [high] | OffchainBook Fee Amount Calculation Error

Description

In `OffchainBook`, `_feeAmountX18` is used to split the given amount to `userAmount` and `feeAmount`.

contracts/OffchainBook.sol SOLIDITY

```solidity
function _feeAmountX18(
    uint64 subaccountId,
    uint32 productId,
    int256 amountX18,
    bool taker
) internal returns (int256, int256) {
    int256 keepRateX18 = ONE -
        fees.getFeeFractionX18(subaccountId, productId, taker);
    int256 newAmountX18 = (amountX18 > 0)
        ? amountX18.mul(keepRateX18)
        : amountX18.div(keepRateX18);
    return (newAmountX18 - amountX18, newAmountX18);
}
```

The fee amount is calculated by `newAmountX18 - amountX18`, always returning a negative fee balance. This leads to inconsistency in total funds since the amount is deducted from users, which is not added anywhere, but instead, is subtracted from the fee account balance.

Proof of Concept

Considering the amountX18 = 100, keepRateX18 = 0.95, it results:

1. newAmountX18 = 95
2. feeAmount = 95 - 100 = -5

Amount 100 is split into 95 and -5, making `newTotal` equal 90, proving an inconsistency.

Remediation

Change the calculation to `amountX18 - newAmountX18`, which works for both positive and negative amounts.

Patch

Calculation changed to `amountX18 - newAmountX18`. Fixed in #116

OS-VTX-ADV-06 [high] | Improper Rounding For BaseSwapped In Swap

Description

In the MathHelper library, the swap function rounds the baseSwapped amount using the sizeIncrement, since the amount needs to be within the baseAtPrice range. The rounds fail to satisfy this condition in the case of a negative amount.

contracts/libraries/MathHelper.sol SOLIDITY

```
int256 baseSwapped;

if (
    (amountSwap > 0 && base + amountSwap > baseAtPrice) ||
    (amountSwap < 0 && base + amountSwap < baseAtPrice)
) {
    // we hit price limits before we exhaust amountSwap
    baseSwapped = baseAtPrice - base;
    baseSwapped -= baseSwapped % sizeIncrement;
} else {
```

Proof of Concept

Let's assume:

1. base = 133, baseAtPrice = 100, sizeIncrement = 10, amountSwap = -40.
2. baseSwappend = 133 - 100 = -33. (equals 33, due to the pool maximum offer amount).
3. Rounding => baseSwapped = -33 - -33%10 = -33 - 7 = -40 (exceeds what the pool can provide).

Remediation

Rounding for negative amounts should be handled.

contracts/libraries/MathHelper.sol SOLIDITY

```
int256 temp = baseSwapped % sizeIncrement;
baseSwapped -= temp + (baseSwapped < 0 && temp != 0 ? sizeIncrement : 0);
```

Patch

Rounding for negative amount handled. Fixed in #116

OS-VTX-ADV-07 [high] | Unreachable DumpFees Function Of OffchainBook

Description

In the OffchainBook contract, dumpFees was used to transfer the funds collected by matching orders to the fee account.

contracts/OffchainBook.sol SOLIDITY

```solidity
function dumpFees() external onlyEndpoint {
    IProductEngine.ProductDelta[]
        memory feeAccDeltas = new IProductEngine.ProductDelta[](1);
    int256 feesAmountX18 = market.collectedFeesX18;
    // https://en.wikipedia.org/wiki/Design_Patterns
    market.collectedFeesX18 = 0;

    // TODO: this is probably fucked for perps
    feeAccDeltas[0] = IProductEngine.ProductDelta({
        productId: QUOTE_PRODUCT_ID,
        subaccountId: FEES_SUBACCOUNT_ID,
        amountDeltaX18: feesAmountX18,
        vQuoteDeltaX18: feesAmountX18
    });
    engine.applyDeltas(feeAccDeltas);
}
```

However, the modifier onlyEndPoint only allows a call from the endpoint contract. Since the call from the endpoint is not created, the function remains unreachable.

Remediation

Add the txType in Endpoint to call the dumpFees function of the Offchainbook, which will be called by the sequencer.

Patch

Added DumpFees transaction type to the EndPoint contract. Fixed in #95

OS-VTX-ADV-08 [high] | No Return Of Funds If DepositCollateral Fails

Description

The submitSlowModeTransaction function in the Endpoint contract takes a specified amount of ERC20 tokens and stores the transaction to slowModeTxs. However, if processSlowModeTransaction fails, the _executeSlowModeTransaction function does not return the funds taken from users. Instead, it silently ignores the issue.

Proof of Concept

Consider this scenario:

1. User submits DepositColleteral transactions, for which the specified amount of tokens is transferred from the user account to the endpoint.
2. If processSlowModeTransaction call fails, _executeSlowModeTransaction is ignoring it silently.

Remediation

In _executeSlowModeTransaction's catch case at the end, if the DepositCollateral transaction failed, then funds should be returned to the user.

Patch

Implemented tryReturnFunds to return the funds. Fixed in #116

OS-VTX-ADV-09 [high] | SocializeSubaccount Covers More Loss Than InsuranceCoverX18

Description

In the `PerpEngine.sol` contract, the `insuranceCoverX18` in the `socializeSubaccount` function has to be of `min` (insurance, loss), rather than be of `max` that was used.

contracts/PerpEngine.sol SOLIDITY

```solidity
if (balance.vQuoteBalanceX18 < 0) {
    int256 insuranceCoverX18 = MathHelper.max(
        insuranceX18,
        -balance.vQuoteBalanceX18
    );
```

Proof of Concept

Assume that the loss is 150 while the insurance is 50. Then, the `insuranceCover` becomes 150. As the insurance is signed int, it goes to -100.

An attacker may cover all of their losses with insurance, although there are not enough in amounts for insurance coverage. This leads to a loss of funds in the protocol.

Remediation

Use the `min` function instead of the `max` function.

Patch

the `max` function changed to the `min` function. Fixed in #116

OS-VTX-ADV-10 [high] | Incorrect BorrowRate Updating In UpdateStates

Description

In SpotEngineStates|'s \spverb|_updateState| function, for \spverb|utilRatio > interestInflection|, the equation is acting improperly in certain cases.

contracts/SpotEngineStates.sol SOLIDITY

```solidity
borrowerRateX18 +=
    config.interestSmallCapX18 +
    config.interestLargeCapX18.mul(
        (
            (ONE - utilizationRatioX18).div(
                ONE - config.interestInflectionUtilX18
            )
        )
    );
```

$$borrowRate = interestFloor + interestSmallCap + interestLargeCap \times \frac{1 - utilRatio}{1 - interestInflection}$$

Since utilRatio = $\frac{borrow}{deposit}$ may equal greater than one, it will decrease the borrowRate. Also, if utilRatio equals to two or three, the borrowRate may become negative, giving profit for the borrowers in reverse for borrowing money.

Remediation

Change the calculation to ensure that interest (borrowRate) increases with the utilization ratio.

Patch

borrowRate calculation changed. Fixed in #98

OS-VTX-ADV-11 [high] | Improper Usage Of Variable For State Updating

Description

In `updateStates` of SpotEngineState contract, `quoteState` variable is used instead of the `state` variable.

contracts/SpotEngineState.sol SOLIDITY

```
for (uint32 i = 0; i < productIds.length; i++) {
    uint32 productId = productIds[i];
    State memory state = states[productId];
    _updateState(productId, quoteState, dt);

    if (productId != QUOTE_PRODUCT_ID) {
        LpState memory lpState = lpStates[productId];
        _updateBalance(state, lpState.base, 0);
        _updateBalance(quoteState, lpState.quote, 0);
        lpStates[productId] = lpState;
        states[productId] = state;
    } else {
        quoteState = state;
    }
}
```

Updates will apply to respective states since updates are applied to `quoteState` instead of `state\, leading to unexpected behaviours within the protocol.`

Remediation

Change the `quoteState` variable to the `state` variable in the highlighted line of the code snippet.

Patch

Code was modified to pass the `state` variable to `updateStates`. Fixed in #101

OS-VTX-ADV-12 [med] | Incorrect Math Operations In MathHelper Library

Description

In the `MathHelper Library`, the overflow and underflow checks for `add`, `sub` and `sqrt` fail for signed integers.

In the `add` function, if x equals 40 and y equals -20, then z which equals `x+y` will equal 20. As this value is less than `x`, this equation fails.

contracts/libraries/MathHelper.sol SOLIDITY

```solidity
function add(int256 x, int256 y) internal pure returns (int256 z) {
    require((z = x + y) >= x, "ds-math-add-overflow");
}
```

In the `sub` function, if x equals 40 and y equals -20, then x which equals `x-y` will equal 60. As this value is greater than `x`, this equation fails as well.

contracts/libraries/MathHelper.sol SOLIDITY

```solidity
function sub(int256 x, int256 y) internal pure returns (int256 z) {
    require((z = x - y) <= x, "ds-math-sub-underflow");
}
```

Since the `sqrt` function is taking signed integers, an error needs to result for the square root of negative integers. Instead, it is returning 1 as the output.

contracts/libraries/MathHelper.sol SOLIDITY

```solidity
function sqrt(int256 y) internal pure returns (int256 z) {
    if (y > 3) {
        z = y;
        int256 x = y / 2 + 1;
        while (x < z) {
            z = x;
            x = (y / x + x) / 2;
        }
    } else if (y != 0) {
        z = 1;
    }
}
```

Remediation

Change either the datatype of variables to unsigned or make the presented checks work for signed integers.

Patch

Overflow and underflow checks have been modified to work for signed integers. Fixed in #96

OS-VTX-ADV-13 [med] | Slow Mode Transactions With Sender Value Allowed As Endpoint

Description

In the Endpoint contract, processSlowModeTransaction allows the transaction in a special case other than msg.sender == txn.sender. As it uses the validateSender function, allowing txnSender == address(this).

contracts/EndPoint.sol SOLIDITY

```solidity
function validateSender(address txSender, address sender) internal view {
    require(
        txSender == sender || txSender == address(this),
        "cannot send slow mode transaction for another address"
    );
}
```

Any user can send a transaction in which the sender is kept to the address (endpoint), which was accepted by processSlowModeTransaction and then is executed on behalf of the endpoint.

To optimize storage in the protocol, it is recommended to reject transactions in the submitSlowModeTransaction function if the sender is not equal to the msg.sender. This is better than checking for this condition in the processSlowModeTransaction function and rejecting the transaction there. Rejecting the transaction in submitSlowModeTransaction also avoids the need to store transactions that were rejected by processSlowModeTransaction.

Proof of Concept

Submit a slowModeTx to the submitSlowModeTransaction, by keeping the sender in the struct as address (endpoint) instead of our address. It is accepted and executed by the processSlowModeTransaction.

Remediation

Check the sender in the transaction struct in submitSlowModeTransaction and reject it there only, making sure that the check was not affected by the validateSender function's special condition of txnSender == address(this).

Patch

validateSender was modified to check the sender variable. Fixed in #116

OS-VTX-ADV-14 [med] | DumpFees Of OffchainBook Fails To Transfer Funds

Description

In the OffchainBook contract, dumpFees transfer the fees collected from matching orders to the engine. Since every Offchainbook is linked to each engine, it applies the deltas onto that engine.

contracts/OffchainBook.sol SOLIDITY

```solidity
function dumpFees() external onlyEndpoint {
    feeAccDeltas[0] = IProductEngine.ProductDelta({
        productId: QUOTE_PRODUCT_ID,
        subaccountId: FEES_SUBACCOUNT_ID,
        amountDeltaX18: feesAmountX18,
        vQuoteDeltaX18: feesAmountX18
    });
    engine.applyDeltas(feeAccDeltas);
}
```

Since the product delta is using QUOTE_PRODUCT_ID, which does not exist in PerpEngine, the transfer is skipped by the PerpEngines.

Remediation

In the case of PerpEngine, the fee must be transferred to the respective productId's vQuoteBalance of FEE_ACCOUNT.

Patch

The fee was transferred to productId's vQuoteBalance of FEE_ACCOUNT in the case of PerpEngine. Fixed in #116

OS-VTX-ADV-15 [med] | Incorrect Cumulative Value Used To Update Perp Lp-State

Description

In the PerpEngineState contract, updateStates updates the base balance of lpState (pool) by using the updateBalance function, for which it needs a lastCumulativeFundingX18 so that lpState holds a lastCumulativeFundingX18 value.

contracts/PerpEngineState.sol SOLIDITY

```solidity
Balance memory balance = Balance({
    amountX18: lpState.base.fromInt(),
    vQuoteBalanceX18: 0,
    lastCumulativeFundingX18: state.cumulativeFundingLongX18
});
_updateBalance(state, balance, 0, 0);
if (lpState.supply != 0) {
    lpState.cumulativeFundingPerLpX18 += balance
        .vQuoteBalanceX18
        .div(lpState.supply.fromInt());
}
lpState.lastCumulativeFundingX18 = state
    .cumulativeFundingLongX18;
```

The function is using state.cumulativeFundingLongX18 instead of lpState.lastCumulativeFundingX18, resulting in zero vQuoteBalanceX18 difference if lpState.base > 0, (cumulativeFundingShortX18 - cumulativeFundingLongX18) applied on amount if lpState.base < 0.

Remediation

Use lpState.lastCumulativeFundingX18 instead of state.cumulativeFundingLongX18.

Patch

lpState.lastCumulativeFundingX18 was used. Fixed in #116

OS-VTX-ADV-16 [med] | BurnLp And MintLp Functions Allow Negative Amounts

Description

Both spot and perp engines allow negative amounts for minting and burning. The amounts from transactions are uint256 and are converted to int256 in Clearinghouse. int256(2**255 - 1) becomes -1 in signed number, therefore, an attacker may still pass negative values to the mintLp and burnLp functions.

contracts/Clearinghouse.sol SOLIDITY

```solidity
productToEngine[txn.productId].mintLp(
    txn.productId,
    subaccountId,
    int256(txn.amountBase).fromInt(),
    int256(txn.quoteAmountLow).fromInt(),
    int256(txn.quoteAmountHigh).fromInt()
);
```

contracts/Clearinghouse.sol SOLIDITY

```solidity
productToEngine[txn.productId].burnLp(
    txn.productId,
    subaccountId,
    int256(txn.amount).fromInt()
);
```

In mintLp, with a negative baseAmount, more Lp can be burned than the Lp amount available, by which an attacker can reduce the supply to zero or negative values.

1. When supply becomes zero, users that have lpBalance of that product face the denial of service since the getHealth function fails with zero division error.
2. When supply becomes zero, mintLp's lpBalance calculation fails as well, since it considers base + quote deposited if supply is zero.
3. When supply becomes negative, unexpected things may occur where supply was used.
4. May decrease the openInterests value with this in PerpMarket.

In burnLp, with a negative lpAmount, any amount of liquidity can be minted as burnLp in clearingHouse does not have an initial health check like mintLp. This is because it expects users to only burn their own lpAmount.

Remediation

Since the converted integers from `uint256` may result in negative values, it is preferred to check them explicitly.

Patch

Checks were implemented to ensure that the integers are positive. Fixed in #116

OS-VTX-ADV-17 [med] | Incorrect CumulativeMultiplier User

Description

In the SpotEngineState contract's _updateBalance function, an incorrect cumulativeMultiplier was used to calculate the normalized amount needed to be separated from the state.totalDepositsNormalizedX18/totalBorrowsNormalizedX18.

contracts/SpotEngineState.sol SOLIDITY

```solidity
if (balance.amountX18 > 0) {
    state.totalDepositsNormalizedX18 -= balance.amountX18.div(
        state.cumulativeDepositsMultiplierX18
    );
} else {
    state.totalBorrowsNormalizedX18 += balance.amountX18.div(
        state.cumulativeBorrowsMultiplierX18
    );
}
```

balance.lastCumulativeMultiplierX18 must be used instead of state.cumulativeDepositsMultiplierX18, since the lastCumulativeMultiplierX18 is the one being applied last on the balance, whereas, state.cumulativeDepositsMultiplierX18 is the current cumultive Multiplier.

Proof of Concept

A possible scenario:

1. Deposit 100 at cumMultiplier 1.2 . balance = 100 ; totalNorm = 83.3
2. Withdraw 125 at cumMultiplier 1.5 . balance = 0 ; totalNorm = 16.7 (remaining).

Remediation

Use balance.lastCumulativeMultiplierX18 instead of state.cumulativeDepositsMultiplierX18.

Patch

balance.lastCumulativeMultiplierX18 was used. Fixed in #116

OS-VTX-ADV-18 [med] | Users Impact Over Pool Price

Description

In both engines, `mintLp` is taking the value of the base token as `quoteAmountLowX18` if the supply is zero. As users can set whatever price they want initially, it is not preferred to leave some control over the pricing to users.

contracts/SpotEngineLp.sol SOLIDITY

```solidity
int256 amountQuoteX18 = (lpState.base.amountX18 == 0)
    ? quoteAmountLowX18
    : amountBaseX18
        .mul(lpState.quote.amountX18.div(lpState.base.amountX18))
        .ceil();
```

Proof of Concept

This may be chained with something else. For example, with the previous negative minting issue, the supply can be zero, then it can be minted at the desired price. Therefore, it is not preferred to leave an unexpected advantage to users.

Remediation

Consider the oracle price if the supply is zero.

Patch

Oracle price was considered if supply was zero. Fixed in #116

OS-VTX-ADV-19 [med] | Improper Normalized Amount Updating On State

Description

In SpotEngine's socializeSubaccount function, the negative balance amount is being added to the total borrow, instead of being subtracted.

contracts/SpotEngine.sol SOLIDITY

```solidity
state.totalBorrowsNormalizedX18 -= balance.amountX18.div(
    state.cumulativeBorrowsMultiplierX18
);
```

Since socializeSubAccount is clearing the borrow and adding the loss to all the deposited users, it must subtract from the total borrow. However, it has minus(-) instead of an add (+), as the amountX18 is already negative.

Remediation

Use add(+) symbol instead of subtraction(-), which is adding the balance to the total borrow amount since the amount is already negative.

Patch

The operation was modified to addition. Fixed in #116

OS-VTX-ADV-20 [med] | Incorrect CanSocialize Condition In GetLiquidationStatus

Description

In `clearingHouse` contract's `getLiquidationStatus` function, the updating condition for `canSocialize` appears incorrect. Based on the documentation comments, if the basis liability exists for any `productId`, then `canSocialize` must be false.

contracts/Clearinghouse.sol SOLIDITY

```solidity
canSocialize = canSocialize && (summary.basisAmountX18 != 0);
```

As the condition for updating the `canSocialize` appears to be the opposite, it is becoming false if any of the product `basisAmount == 0`.

Remediation

Change the above condition to use == instead of `!=`.

contracts/Clearinghouse.sol SOLIDITY

```solidity
canSocialize = canSocialize && (summary.basisAmountX18 == 0);
```

Patch

Fixed in #116

OS-VTX-ADV-21 [low] | Improper Update Of MaxHealthGroup

Description

The `registerProductForId` of `clearingHouse` is accepting any new health group number without any restrictions, which updates the `maxHealthGroup` if it is greater than that.

contracts/ClearingHouse.sol SOLIDITY

```solidity
if (healthGroup > maxHealthGroup) {
    maxHealthGroup = healthGroup;
}
```

Since the iteration over all health groups is completed by using `maxHealthGroup`, if a health group with a larger number is inserted, it leads to gaps in the mapping, causing unnecessary gas consumption. If the number is very high in either an intended or unintended manner, it leads to a denial of service, since the iteration is impossible.

Proof of Concept

Add a new product in any engine with a high health group number. Then, it will be accepted by the `registerProductId` function.

Remediation

Add a check in `registerProductId` function if new `healthGroup > maxHealthGroup`. Check `healthGroup == maxHealthGroup + 1`

Patch

`healthGroup == maxHealthGroup + 1` condition was ensured for new `healthGroup` which is greater than `maxHealthGroup`. Fixed in #116

OS-VTX-ADV-22 [low] | Rounding Bug For QuoteSwappedX18 In Swap Function

Description

In the `MathHelper` contract, `swap` is rounding the `quoteSwappedX18` downwards at the end, even if the amount is positive or negative.

contracts/libraries/MathHelper.sol SOLIDITY

```solidity
int256 quoteSwappedX18 = (k / (base + baseSwapped) - quote).fromInt();
if (amountSwap > 0) {
    quoteSwappedX18 = quoteSwappedX18.mul(keepRateX18).toInt();
} else {
    quoteSwappedX18 = quoteSwappedX18.div(keepRateX18).toInt();
}
return (baseSwapped.fromInt(), quoteSwappedX18.toInt().fromInt());
```

It is acceptable for negative amounts since the amount taken from the protocol is reduced. However, for positive amounts, it must be rounded upwards.

Remediation

In the else case from the code above, ceil the value.

Patch

The value was ceiled in the else case. Fixed in #116

OS-VTX-ADV-23 [low] | SafeGuarding Time And Price Inputs From Sequencer

Description

In the `Endpoint` contract, `UpdateTime` and `UpdatePrice` transactions are used to update the price and time. However, there are no checks to ensure the values are safe. The checks that need to be ensured are:

1. Price has to be of non-negative value.
2. Time has to be greater than the last update time. (not equal)

Since both cases lead to unexpected behaviours, with price and time being sensitive values, it is preferred to safeguard them.

Remediation

Ensure required checks are implemented for time and price.

Patch

The required checks were ensured for time and price. Fixed in #116.

OS-VTX-ADV-24 [low] | AddEngine Accepts Zero Address For Engine

Description

In the clearingHouse contract, addEngine is not checking that the given engine address is not a zero address.

contracts/Clearinghouse.sol SOLIDITY

```solidity
function addEngine(address engine, IProductEngine.EngineType engineType)
    external
    onlyOwner
{
    require(address(engineByType[engineType]) == address(0));
    IProductEngine productEngine = IProductEngine(engine);
    // Register
    supportedEngines.push(engineType);
    ...
}
```

Since adding a new engine was blocked by the condition of ‘current address != zero‘, an owner may add a zero address as an engine multiple times, leading to duplicates in the supportedEngines list. Although this may occur accidentally, it is preferred to make the contract handle these cases.

Proof of Concept

When the address of the engine is not set, addEngine with engine address = 0. The supportedEngines list is appended with duplicate entries of EngineType.

Remediation

Add the check in addEngine that engine != address(0).

Patch

Fixed in #116

05 | General Findings

Here we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent antipatterns and could lead to security issues in the future.

ID	Description
OS-VTX-SUG-00	Code redundancy has been observed in several parts of the code.
OS-VTX-SUG-01	The token address should be validated when depositing collateral for a given `productId`.
OS-VTX-SUG-02	An unnecessary external call to `submitSlowModeTransaction` is made from `depositCollateral`.
OS-VTX-SUG-03	The use of Int48 variables is unnecessary for `RiskStore`.
OS-VTX-SUG-04	Invalid or incorrect assignments and parameters are being used or passed.
OS-VTX-SUG-05	Access to the endpoint should be frozen until both engines are added.
OS-VTX-SUG-06	The `productId` should be validated when executing the withdraw collateral and deposit collateral functions.
OS-VTX-SUG-07	There are rounding issues in the `burnLp` function in `SpotEngineLp.sol`.
OS-VTX-SUG-08	A strict condition for base and quote values is not necessary in the `ammEquilibrium` function.
OS-VTX-SUG-09	Checks are missing in the `liquidateSubaccount` function to verify whether it is complete in SPREAD mode.
OS-VTX-SUG-10	The initialization of an engine does not trigger any event emissions.
OS-VTX-SUG-11	When executing a `slowModeTransaction`, it is advisable to validate the count.
OS-VTX-SUG-12	To optimize gas usage, the code for calculating an account's health should be structured appropriately.

OS-VTX-SUG-13	The `assertLiquidationAmount` function contains unnecessary checks.
OS-VTX-SUG-14	To save gas, it is recommended to use the `!` = operator instead of the `abs()` function.
OS-VTX-SUG-15	WWhen converting to `toInt()` in the `SpotEngineLp.sol` swap function, the decimals are truncated.
OS-VTX-SUG-16	Gas consumption can be reduced by utilizing clones to deploy `offchainbooks`.

OS-VTX-SUG-00 | Code Redundancy

Description

Redundant assignment of `balance.lastCumulativeFundingX18` in `_updateBalance`, since the value was overridden below.

contracts/PerpEngineState.sol SOLIDITY

```solidity
  int256 diffX18 = cumulativeFundingAmountX18 -
      balance.lastCumulativeFundingX18;
    int256 deltaQuoteX18 = vQuoteDeltaX18 -
  ↪ diffX18.mul(balance.amountX18);
    balance.lastCumulativeFundingX18 = cumulativeFundingAmountX18;

    // apply delta
    balance.amountX18 += balanceDeltaX18;

    // apply vquote
    balance.vQuoteBalanceX18 += deltaQuoteX18;

    // post update
    if (balance.amountX18 > 0) {
        state.openInterestX18 += balance.amountX18;
        balance.lastCumulativeFundingX18 =
  ↪ state.cumulativeFundingLongX18;
    } else {
        balance.lastCumulativeFundingX18 =
  ↪ state.cumulativeFundingShortX18;
    }
```

In the `submitTransactions` function, a part of the code seems to be redundant. It can be replaced with `fSubmitTransactions` function call.

SOLIDITY

```solidity
  function fSubmitTransactions(bytes[] calldata transactions) external
  ↪ {
    require(
        msg.sender == sequencer,
         "Only the sequencer can submit transactions"
     );
    require(idx == nSubmissions, "Invalid submission index");
    for (uint256 i = 0; i < transactions.length; i++) {
        bytes calldata transaction = transactions[i];
        processTransaction(transaction);
    }
    nSubmissions += uint64(transactions.length);
    emit SubmitTransactions(transactions);
 }
```

The PNL type of healthType is never used anywhere in the code.

SOLIDITY

```
    ) internal pure returns (int256) {
        // (1 + imf * sqrt(amount))
        if (healthType == IProductEngine.HealthType.PNL) {
            return ONE;
        }

        // TODO: skip if possible; sqrt is expensive
```

Unnecessary definition of the variable `_clearinghouse` in `processTransaction` function, since the variable `clearinghouse` is already initiated as `IClearinghouse`.

contracts/Endpoint.sol SOLIDITY

```
  else if (txType == TransactionType.UpdateTime) {
            UpdateTime memory txn = abi.decode(transaction[1:],
    ↪   (UpdateTime));
            time = txn.time;
            IClearinghouse _clearinghouse = clearinghouse;
            IProductEngine.EngineType[] memory engineTypes =
    ↪   _clearinghouse
                .getSupportedEngines();
```

The variable `_endpoint` in the `BaseEngine` abstract is unused.

contracts/BaseEngine.sol SOLIDITY

```
abstract contract BaseEngine is IProductEngine, EndpointGated {
    using PRBMathSD59x18 for int256;
    IEndpoint internal _endpoint;
    IClearinghouse internal _clearinghouse;
    IFeeCalculator internal _fees;
    uint32[] internal productIds;
```

Code redundancy in `liquidateSubaccount` function.

contracts/Clearinghouse.sol SOLIDITY

```solidity
        if (
            getHealthX18(txn.liquidateeId,
     ↪  IProductEngine.HealthType.INITIAL) >=
            0
        ) {
            return;
        }
```

Unwanted wrapping for`amountLpX18` in `burnLp` for conversion.

contracts/SpotEngineLp.sol SOLIDITY

```solidity
        State memory quote = states[QUOTE_PRODUCT_ID];

        int256 amountLpX18 = int256(amountLpX18);
        if (amountLpX18 == type(int256).max) {
```

It is not required to explicitly wrap the `PRBMathSD59x18`, since `PRBMathSD59x18` was used for int256.

contracts/Clearinghouse.sol SOLIDITY

```solidity
bool isLiability = false;
        int256 amountToLiquidateX18 = PRBMathSD59x18.fromInt(txn.amount);
        LiquidationVars memory vars;
```

contracts/OffchainBook.sol SOLIDITY

```solidity
        takerAmountDeltaX18 = PRBMathSD59x18.fromInt(takerAmountDelta);
        int256 makerQuoteDeltaX18 = PRBMathSD59x18.mul(
            takerAmountDeltaX18,
            maker.priceX18
        );
```

An unnamed parameter is unused in the `BaseEngine.sol` initialize function.

SOLIDITY

```solidity
    function _initialize(
        address _clearinghouseAddr,
        address,
        address _endpointAddr,
        address _admin,
        address _feeAddr
    ) internal initializer {
```

Remediation

Remove or replace the redundant lines of code.

OS-VTX-SUG-01 | Validate Token Address

Description

While depositing or withdrawing collateral for a `productId`, ensure to validate the token address returned from the spot engine for that `productId`.

Remediation

Add a check to ensure that the token address is non-zero.

DIFF

```diff
--- a/contracts/Endpoint.sol
+++ b/contracts/Endpoint.sol
@@ -146,6 +146,7 @@ contract Endpoint is IEndpoint, EIP712Upgradeable,
     ↪  OwnableUpgradeable {
             IERC20Base token = IERC20Base(
                 spotEngine.getConfig(txn.productId).token
             );
+            require(address(token) != address(0), "Error message here");
             handleDepositTransfer(token, sender, txn.amount);
         }
```

OS-VTX-SUG-02 | Unnecessary External Call

Description

There is no need to make an unnecessary external call to the function `submitSlowModeTransaction` in the `depositCollateral` function. A simple internal function call is sufficient since the `msg.sender` does not change.

Remediation

Replace the external call with an internal function.

contracts/Endpoint.sol SOLIDITY

```solidity
        bytes memory encodedTx = abi.encode(txn);
        bytes memory transaction = abi.encodePacked(
            uint8(TransactionType.DepositCollateral),
            encodedTx
        );

        submitSlowModeTransaction(transaction);
```

OS-VTX-SUG-03 | Unnecessary Long Integers

Description

The values in the RiskStore of the clearing house range from 1e9 to 2*1e9. Therefore, 32-byte integers are sufficient rather than 48-byte integers.

Remediation

Use int32 for the variables in the RiskStore instead of int48.

DIFF

```diff
--- a/contracts/interfaces/clearinghouse/IClearinghouseState.sol
+++ b/contracts/interfaces/clearinghouse/IClearinghouseState.sol
@@ -10,11 +10,11 @@ interface IClearinghouseState {
         // between 0 and 2
         // these integers are the real
         // weights times 1e9
-        int48 longWeightInitial;
-        int48 shortWeightInitial;
-        int48 longWeightMaintenance;
-        int48 shortWeightMaintenance;
-        int48 largePositionPenalty;
+        int32 longWeightInitial;
+        int32 shortWeightInitial;
+        int32 longWeightMaintenance;
+        int32 shortWeightMaintenance;
+        int32 largePositionPenalty;
     }
```

OS-VTX-SUG-04 | Invalid Code

Description

Incorrect value of the constant FUNDING_PERIOD_X18.

contracts/PrepEngineLp.sol SOLIDITY

```solidity
int256 constant EMA_TIME_CONSTANT_X18 = 998334721450938752;
int256 constant FUNDING_PERIOD_X18 = 86000_000000000000000000;
```

In the functions matchOrderAmm, matchOrders, and swapAmm, while retrieving the fee from the function _feeAmountX18 for taker, it is required to pass the taker bool flag as true instead of false.

contracts/OffchainBook.sol SOLIDITY

```solidity
        (takerFeeX18, takerQuoteDeltaX18) = _feeAmountX18(
            takerSubaccountId,
            _market.productId,
            takerQuoteDeltaX18,
            false
        );
```

contracts/OffchainBook.sol SOLIDITY

```solidity
        int256 takerFeeX18;
        (takerFeeX18, takerQuoteDeltaX18) = _feeAmountX18(
            takerSubaccountId,
            _market.productId,
            takerQuoteDeltaX18,
            false
        );
```

contracts/OffchainBook.sol SOLIDITY

```solidity
        int256 takerFeeX18;
        (takerFeeX18, takerQuoteDeltaX18) = _feeAmountX18(
            ordersInfo.takerSubaccountId,
            _market.productId,
            takerQuoteDeltaX18,
            false
        );
```

Remediation

Replace the invalid lines of code.

OS-VTX-SUG-05 | Restrict Endpoint Access

Description

If both engines are not initialized via `Clearinghouse.sol`, the transactions to `Endpoint.sol` are likely to fail. Therefore, it is preferred to freeze the contract until both engines are properly initialized.

Remediation

Add a function in the `Clearinghouse.sol` which returns the length of `engineByType` (returns the number of engines). Then, add a modifier to the `Endpoint.sol` that validates the fact that both engines are initialized via the `Clearinghouse.sol`. Lastly, add this modifier to every function in the `Endpoint.sol` contract.

OS-VTX-SUG-06 | Validate ProductId

Description

In the functions withdrawCollateral and depositCollateral in the Clearinghouse.sol contract, it is better to validate if the txn.productId is in the spot engine or not, before depositing or withdrawing.

Remediation

Add checks to ensure that the given productId is a valid product ID from the spot engine.

OS-VTX-SUG-07 | Rounding Issues

Description

The rounding error occurs in the `burnLp` function of `SpotEngineLp.sol` due to an incorrect sequence of arithmetic operations when calculating `amountBaseX18` and `amountQuoteX18`.

```solidity
int256 amountLp = amountLpX18.toInt();

int256 amountBaseX18 = (
    MathHelper.mul(amountLp, lpState.base.amountX18 / lpState.supply)
);
int256 amountQuoteX18 = (
    MathHelper.mul(amountLp, lpState.quote.amountX18 / lpState.supply)
);

_updateBalance(base, lpState.base, -amountBaseX18);
_updateBalance(quote, lpState.quote, -amountQuoteX18);
lpState.supply -= amountLp
```

Remediation

Rewrite the equation to first do the multiplication, and then division.

OS-VTX-SUG-08 | Unrequired Strict Condition

Description

The function ammEquilibrium should have to return (0,0) if one of the values from the base or quote is zero. Therefore, strict condition is not required.

```solidity
  ) internal pure returns (int256, int256) {
    if (baseX18 == 0 && quoteX18 == 0) {
      return (0, 0);
    }
    int256 k = baseX18.toInt() * quoteX18.toInt();
```

Remediation

Replace the && operation with || operation.

OS-VTX-SUG-09 | Missing Checks In Liquidate In Spread Mode

Description

While liquidating a subaccount via `liquidateSubaccount` in SPREAD, checks for the provided health groups are missing to validate that the `healthGroup` has both spot and perp products.

```solidity
isLiability = summary.basisAmountX18 < 0;

          HealthGroup memory healthGroup = healthGroups[txn.healthGroup];

          vars.liquidationPriceX18 = getSpreadLiqPriceX18(
```

Remediation

Add checks to the health group to verify that it has both spot and perp products.

```solidity
isLiability = summary.basisAmountX18 < 0;

          HealthGroup memory healthGroup = healthGroups[txn.healthGroup];
          require(healthGroup.spotId != 0 && healthGroup.perpId != 0);
          vars.liquidationPriceX18 = getSpreadLiqPriceX18(
```

OS-VTX-SUG-10 | Missing Events

Description

When an engine is initialized as either a perp or a spot, it is better to emit events.

Remediation

Add new events and emit them when an engine is initialized via `addEngine` function in `Clearinghouse.sol`

OS-VTX-SUG-11 | Unbounded Transaction Execution

Description

In the endpoint contract, while executing a slowMode transaction via the executeSlowModeTransactions function, it is better to ensure that the provided count is <= than the length of the slowModeTxs array.

Remediation

Add a required statement to ensure that count is less or equal to the length of slowModeTxs.

DIFF

```diff
--- a/contracts/Endpoint.sol
+++ b/contracts/Endpoint.sol
@@ -188,6 +188,7 @@ contract Endpoint is IEndpoint, EIP712Upgradeable,
     ↪  OwnableUpgradeable {

     function executeSlowModeTransactions(uint32 count) external {
         SlowModeConfig memory _slowModeConfig = slowModeConfig;
+        require(count <= (_slowModeConfig.txCount -
     ↪  _slowModeConfig.txUpTo), "Error message here");
         while (count > 0) {
             _executeSlowModeTransaction(_slowModeConfig, false);
             --count;
```

OS-VTX-SUG-12 | Improper Code Structure

Description

To optimize gas usage, it is recommended to structure the code in the `_getHealth` function in `Clearinghouse.sol` such that risks and penalties are applied only if spot/perp products exist in the health group. However, this should not be done every time, as the `sqrt` function is costly and can reduce gas usage.

Remediation

Move the code which calculates risks into the respective conditioned block, so that the calculations are only done if spot/perp ID's exists.

DIFF

```diff
--- a/contracts/Clearinghouse.sol
+++ b/contracts/Clearinghouse.sol
@@ -253,38 +253,45 @@ contract Clearinghouse is ClearinghouseRisk,
    ↪  IClearinghouse {
             }

             // apply risk for spot and perp positions
-            int256 combinedSpotX18 = healthVars.spotAmountX18 +
-                healthVars.spotInLpAmountX18;
-            healthX18 += RiskHelper
-                ._getWeightX18(healthVars.spotRisk, combinedSpotX18,
    ↪  healthType)
-                .mul(combinedSpotX18)
-                .mul(healthVars.spotPriceX18);
-
-            int256 combinedPerpX18 = healthVars.perpAmountX18 +
-                healthVars.perpInLpAmountX18;
-            healthX18 += RiskHelper
-                ._getWeightX18(healthVars.perpRisk, combinedPerpX18,
    ↪  healthType)
-                .mul(combinedPerpX18)
-                .mul(healthVars.perpPriceX18);
+            if (group.spotId != 0){
+                int256 combinedSpotX18 = healthVars.spotAmountX18 +
+                    healthVars.spotInLpAmountX18;
+                healthX18 += RiskHelper
+                    ._getWeightX18(healthVars.spotRisk, combinedSpotX18,
    ↪  healthType)
+                    .mul(combinedSpotX18)
+                    .mul(healthVars.spotPriceX18);
+           }
+           if (group.perpId != 0){
+
+               int256 combinedPerpX18 = healthVars.perpAmountX18 +
+                   healthVars.perpInLpAmountX18;
+               healthX18 += RiskHelper
+                   ._getWeightX18(healthVars.perpRisk, combinedPerpX18,
    ↪   healthType)
+                   .mul(combinedPerpX18)
+                   .mul(healthVars.perpPriceX18);
+           }

            // apply penalties on amount in LPs
-           healthX18 -= (ONE -
-               RiskHelper._getWeightX18(
-                   healthVars.spotRisk,
-                   healthVars.spotInLpAmountX18,
-                   healthType
-               )).mul(healthVars.spotInLpAmountX18).mul(
-                   healthVars.spotPriceX18
-               );
-
-           healthX18 -= (ONE -
-               RiskHelper._getWeightX18(
-                   healthVars.perpRisk,
-                   healthVars.perpInLpAmountX18,
-                   healthType
-               )).mul(healthVars.perpInLpAmountX18).mul(
-                   healthVars.perpPriceX18
-               );
+           if(group.spotId != 0){
+               healthX18 -= (ONE -
+                   RiskHelper._getWeightX18(
+                       healthVars.spotRisk,
+                       healthVars.spotInLpAmountX18,
+                       healthType
+                   )).mul(healthVars.spotInLpAmountX18).mul(
+                       healthVars.spotPriceX18
+                   );
+           }
+           if(group.perpId != 0){
+               healthX18 -= (ONE -
+                   RiskHelper._getWeightX18(
+                       healthVars.perpRisk,
+                       healthVars.perpInLpAmountX18,
+                        healthType
+                    )).mul(healthVars.perpInLpAmountX18).mul(
+                        healthVars.perpPriceX18
+                    );
+            }
         }
     }
```

OS-VTX-SUG-13 | Unrequired Checks With Liquidation Amount

Description

In the `assertLiquidationAmount` function, the checks for `originalBalanceX18` are unrequired, since the conditions are validated in the checks. For when `liquidationAmountX18` > 0 and `originalBalanceX18` >= `liquidationAmountX18`, these checks ensure that `originalBalanceX18` > 0, therefore the extra checks are not required.

SOLIDITY

```solidity
        if (liquidationAmountX18 > 0) {
            require(
                originalBalanceX18 >= liquidationAmountX18 &&
                    originalBalanceX18 > 0,
                ERR_NOT_LIQUIDATABLE_AMT
            );
        } else {
            require(
                originalBalanceX18 <= liquidationAmountX18 &&
                    originalBalanceX18 < 0,
                ERR_NOT_LIQUIDATABLE_AMT
            );
```

Remediation

Remove the extra and unnecessary checks.

OS-VTX-SUG-14 | Lower Gas Check

Description

In the function `settlePnl` to validate the `canSettleX18.abs() > 0`, it is better to use `!=`, since the `abs()` uses higher gas than a single opcode for `!=`.

SOLIDITY

```solidity
            ) = getSettlementState(productId, subaccountId);

            if (canSettleX18.abs() > 0) {
                // Product and balance updates in getSettlementState
                state.availableSettleX18 -= canSettleX18;
                balance.vQuoteBalanceX18 -= canSettleX18;
```

Remediation

Replace the check with `canSettleX18 != 0`.

OS-VTX-SUG-15 | Rounding Issue While Swapping

Description

The swap function in SpotEngineLp.sol currently uses mathhelper to calculate the swap amount, which takes base and quote values without decimal precision (not X18). These values are then converted to X18 during calculations in the swap function, which can result in a lack of rounding of the original amount. To address this issue, it is recommended to rewrite the swap function to take X18 integers instead.

Remediation

Rewrite the swap function to take the X18 base and quote amounts.

DIFF

```diff
--- a/contracts/libraries/MathHelper.sol
+++ b/contracts/libraries/MathHelper.sol
@@ -107,7 +107,7 @@ library MathHelper {
         if (base == 0 || quote == 0) {
             return (0, 0);
         }
-        int256 currentPriceX18 = quote.fromInt().div(base.fromInt());
+        int256 currentPriceX18 = quote.div(base);
 
         int256 keepRateX18 = 1e18 - lpSpreadX18;
 
@@ -150,6 +150,6 @@ library MathHelper {
         } else {
             quoteSwappedX18 = quoteSwappedX18.div(keepRateX18).toInt();
         }
-        return (baseSwapped.fromInt(),
     ↪  quoteSwappedX18.toInt().fromInt());
+        return (baseSwapped.fromInt(), quoteSwappedX18.fromInt());
     }
 }
```

OS-VTX-SUG-16 | Offchainbook Clones

Description

To reduce gas usage, it is recommended to minimize the use of clones when deploying off-chain books. By using clones sparingly, it becomes easier to interact with a principal logic implementation since the logic of all `offchainbooks` are the same.

More about minimal clones: docs.openzeppelin.com/contracts/4.x/api/proxy##minimal_clones

Remediation

Use minimal clones while deploying the `offchainbooks`.

A Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings can be found in the General Findings section.

Critical

Vulnerabilities that immediately lead to loss of user funds with minimal preconditions

Examples:

- Misconfigured authority or access control validation
- Improperly designed economic incentives leading to loss of funds

High

Vulnerabilities that could lead to loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions
- Exploitation involving high capital requirement with respect to payout

Medium

Vulnerabilities that could lead to denial of service scenarios or degraded usability.

Examples:

- Malicious input that causes computational limit exhaustion
- Forced exceptions in normal user flow

Low

Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions

Informational

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants
- Improved input validation