SALUS SECURITY

SEP 2024

CODE SECURITY ASSESSMENT

LOGX

Overview

Project Summary

- Name: LogX - Trading
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/eugenix-io/logx-trading-contracts
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	LogX - Trading
Version	v3
Type	Solidity
Dates	Sep 07 2024
Logs	Sep 01 2024, Sep 06 2024; Sep 07 2024

Vulnerability Summary

Total High-Severity issues	4
Total Medium-Severity issues	13
Total Low-Severity issues	2
Total informational issues	3
Total	22

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Lack of health check after collateral withdrawal	High	Business Logic	Resolved
2	Incorrect funding fee calculation	High	Business Logic	Resolved
3	Missing the implementation of isHealthy function	High	Business Logic	Resolved
4	Lack of signature verify in SettleSubaccountPnl transaction	High	Business Logic	Resolved
5	Incorrect availableSettle calculation	Medium	Business Logic	Resolved
6	Perform account health check prematurely	Medium	Business Logic	Resolved
7	Centralization risk	Medium	Centralization	Mitigated
8	Missing order expiration check	Medium	Business Logic	Resolved
9	Non-functional insurance in Clearinghouse	Medium	Business Logic	Resolved
10	Lack of startTime validation check in stakingLogX	Medium	Centralization	Resolved
11	stLogX may be cleared in liquidation process	Medium	Business Logic	Resolved
12	Lack of bridgeOutContract verification in withdrawLogX	Medium	Centralization	Resolved
13	Lack of stakerContract verification in claimRewards	Medium	Centralization	Resolved
14	Lack of productId verification in stakeLogX	Medium	Centralization	Resolved
15	Lack of tokenAmount verification in claimLogX	Medium	Centralization	Resolved
16	Lack of productId verification in claimRewards	Medium	Centralization	Resolved
17	Lack of tokenAmount verification in stakeLogX	Medium	Centralization	Resolved
18	Missing minSize order size check	Low	Business Logic	Resolved
19	Some orders will never be fully filled	Low	Business Logic	Resolved

20	Missing gap	Informational	Code Quality	Resolved
21	The error message is not specific	Informational	Code Quality	Acknowledged
22	The same order can only be filled once	Informational	Data Validation	Resolved

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Lack of health check after collateral withdrawal

Severity: High	Category: Business Logic
Target: - src/Clearinghouse.sol	

Description

In the Clearinghouse contract, the `withdrawLogX` and `withdrawCollateral` functions do not include a health check on the account following a withdrawal. This oversight permits users to withdraw all collateral even when their positions are underperforming, which could result in bad debt for the protocol.

src/Clearinghouse.sol:L215-L234

```
function withdrawLogX(WithdrawLogX memory withdrawRequest) external onlyEndpoint {
    ...
    IHyperlane(withdrawRequest.bridgeOutContract).transferRemote{value:
uint128(amountAfterFees)}(uint32(withdrawRequest.destinationChainId),receiver,
uint128(amountAfterFees));
    ...
}
```

src/Clearinghouse.sol:L236-L255

```
function withdrawCollateral(WithdrawCollateral memory withdrawCollateralRequest)
external onlyEndpoint {
    ...
    IHyperlane(token).transferRemote(chainId, receipientInBytes32,
uint128(amountAfterFees));
    ...
}
```

Recommendation

At the end of the `withdrawLogX` and `withdrawCollateral` functions, make sure the account is healthy.

Status

This issue has been resolved by the team via off-chain check.

2. Incorrect funding fee calculation

Severity: High	Category: Business Logic

Target:
- src/PerpEngine.sol
- src/PerpEngineState.sol

Description

When the `insurance` fund cannot fully cover user losses, the shortfall is distributed among other market participants, causing the cumulativeFunding to increase the `fundingPerShare`. The `fundingPerShare` reflects the loss each unit of position size must bear. However, during user settlement, the FundingFee is mistakenly calculated by multiplying fundingPerShare by the position value instead of the position size, resulting in an incorrect FundingFee. This may result in unexpectedly overcharged or undercharged fees.

src/PerpEngine.sol:L140-L177

```
function socializeSubaccount(bytes32 subaccount, int128 insurance, int128 pnl, uint32
productId)
        external
        returns (int128)
{
    ...
    int128 netStateOpenInterest = state.longOpenInterest +
state.shortOpenInterest.abs();
    int128 fundingPerShare = -pnl.div(netStateOpenInterest);
    state.cumulativeFundingLongX18 += fundingPerShare;
    state.cumulativeFundingShortX18 -= fundingPerShare;
    ...
}
```

src/PerpEngineState.sol:L47-L52

```
function _updateBalance(
     State memory state,
     Balance memory balance,
     int128 balanceDelta,//amount is in tokens
     int128 vQuoteDelta//this is in USD
) internal pure returns (int128 fundingFees, int128 realisedPnl){
    ...
    int128 diffX18 = cumulativeFundingAmountX18 -
        balance.lastCumulativeFundingX18;
    fundingFees = -diffX18.mul(balance.vQuoteBalance);
    ...
}
```

Recommendation

When calculating the settlement fee, use the `balance.amount` instead of the `balance.vQuoteBalance`.

Status

This issue has been resolved by the team with commit 9e644ec.

3. Missing the implementation of isHealthy function

Severity: High	Category: Business Logic
Target: - src/OffChainExchange.sol	

Description

The OffChainExchange contract needs to call the `isHealthy` function for account health checks after executing `matchOrders`, but the function is not implemented in the contract.This could lead to accounts being in a liquidation state due to a decrease in their health value after order matching.

src/OffChainExchange.sol:L310-L314

```
function isHealthy(
    bytes32 /* subaccount */
) internal view virtual returns (bool) {
    return true;
}
```

Recommendation

Correctly implement the `isHealthy` function to check the account's health value, and ensure that for spot trading, the account's spot balance cannot be less than 0.

Status

This issue has been resolved by the team via off-chain check.

4. Lack of signature verify in SettleSubaccountPnl transaction

Severity: High	Category: Business Logic

Target:
- src/Endpoint.sol

Description

If traders have some negative Pnl in the trading process, traders can use existing other collaterals to cover these negative Pnl via `SettleSubaccountPnl` transaction.

The issue arises from the lack of verification of the trader's signature. Malicious users can exploit this vulnerability by triggering the victim's `SettleSubaccountPnl` transaction to manipulate the victim's nonce. As a result, the victim's signature may continuously fail due to an invalid nonce.

src/Endpoint.sol:L159-L165

```
function processTransaction(bytes calldata transaction, bytes calldata signature, bytes
calldata signature2) internal {
    TransactionType txType = TransactionType(uint8(transaction[0]));
    ...
    if (txType == TransactionType.SettleSubaccountPnl) {
            SettleSubaccountPnl memory request =
abi.decode(transaction[1:],(SettleSubaccountPnl));
            validateAccountDetails(request.subAccountId, request.sessionKey,
request.nonce);
            clearinghouse.settleSubaccountPnl(request.subAccountId,
request.spotProductIds, request.spotPricesX18);
        }
    ...
}
```

Recommendation

Verify users' signature in SettleSubaccountPnl transaction.

Status

This issue has been resolved by the team. The team states that only the system can call this transaction.

5. Incorrect availableSettle calculation

Severity: Medium	Category: Business Logic
Target: - src/Clearinghouse.sol	

Description

When using the spot market to settle user losses, if the value of the spot product is sufficient to cover the loss, the required `amount` will be calculated and deducted from the user's balance. However, when calling `updateAvailableSettle` to set `availableSettle`, it incorrectly deducted `balance.amount` instead of adding the calculated `amount`.

src/Clearinghouse.sol:L490-L497

```
function _settleLiqPnlUsingSpots(
    IEndpoint.LiquidateSubaccount calldata txn,
    ISpotEngine spotEngine,
    int128 totalPnl
) private returns (int128) {
    ...
    if (balance.amount.mul(txn.spotPricesX18[i]) < -totalPnl) {
        ...
    } else {
        // If yes, then update the balance to pay the PnL
        int128 amount = totalPnl.div(txn.spotPricesX18[i]);
        spotEngine.updateBalance(spotProductIds[i], liquidatee, amount);
        spotEngine.updateAvailableSettle(spotProductIds[i], -balance.amount);
        totalPnl = 0;
        break;
    }
    ...
}
```

Recommendation

Update available settle correctly.

Status

This issue has been resolved by the team with commit 5bd26c9.

6. Perform account health check prematurely

Severity: Medium	Category: Business Logic
Target: - src/OffChainExchange.sol	

Description

During the order matching process in the `matchOrders` function, the account's health status is checked after executing the orders. However, after this check, the protocol still charges fees to the account. This could lead to a situation where, after the fees are charged, the user's health value may fall below the protocol's requirements, potentially resulting in a liquidation state or causing the spot market balance to become negative.

src/OffChainExchange.sol:L402-L411

```
function matchOrders(IEndpoint.MatchOrdersWithSigner calldata txn)
    external
    onlyEndpoint
{
    ...
    require(isHealthy(taker.subAccountId), ERR_INVALID_MAKER);
    require(isHealthy(maker.subAccountId), ERR_INVALID_MAKER);
    if (feesAndPnl.makerFee != 0) {
        collectFees(feesAndPnl.makerFee, maker.subAccountId, TRADING_FEES_ACCOUNT);
    }
    if (feesAndPnl.takerFee != 0) {
        collectFees(feesAndPnl.takerFee, taker.subAccountId, TRADING_FEES_ACCOUNT);
    }
    ...
}
```

Recommendation

Check the account's health status after charging fees.

Status

This issue has been resolved by the team via off-chain check.

7. Centralization risk

Severity: Medium	Category: Centralization
Target: - src/Endpoint.sol - src/PerpEngineState.sol - src/SpotEngineState.sol - src/Clearinghouse.sol	

Description

In Endpoint and other contracts, there exists some privileged owner roles, for example, owner, sequencer and rebalancer. These roles have authority over key operations such as updating critical parameters, submitting transactions, and rebalancing funds.

If these roles' private keys were compromised, an attacker could exploit this access to withdraw all tokens.

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

This issue has been mitigated by the team. The team has already transferred the ownership of the Logx token to a multi-sig account during the 0xd69f7d... transaction and stated that the privileged addresses of the remaining contracts will be transferred to the multisig account after the official deployment.

8. Missing order expiration check

Severity: Medium	Category: Business Logic
Target: - src/OffChainExchange.sol	

Description

When traders submit an order, they can specify an expiration time. If the order is not matched by the expiration date, it should be automatically canceled.

The problem is that we miss checking the order's expiration.

src/OffChainExchange.sol:L181-L184

```
function _expired(uint64 expiration) internal view returns (bool) {
    return expiration <= block.timestamp;
}
```

Recommendation

Check the order's expiration when we match the order.

Status

This issue has been resolved by the team via off-chain check.

9. Non-functional insurance in Clearinghouse

Severity: Medium	Category: Business Logic
Target: - src/Clearinghouse.sol	

Description

In LogX, bad debt may arise if traders' positions are liquidated. The insurance fund is used to cover a portion of this bad debt.

The issue is that the insurance can only be deposited through the Endpoint. However, the Endpoint contract lacks a function to trigger the `depositInsurance` function.

src/OffChainExchange.sol:L307-L315

```
function depositInsurance(IEndpoint.DepositInsurance calldata txn)
    external
    virtual
    onlyEndpoint
{
    require(txn.amount <= INT128_MAX, ERR_CONVERSION_OVERFLOW);
    int128 amount = int128(txn.amount);
    insurance += amount;
}
```

Recommendation

If an insurance feature is needed, support depositing insurance via the Endpoint contract.

Status

This issue has been resolved by the team with commit 9e644ec.

10. Lack of startTime validation check in stakeLogX

Severity: Medium	Category: Centralization
Target: - src/Endpoint.sol	

Description

In LogX, traders can stake their LogX tokens to earn staking rewards. However, the staked tokens cannot be unlocked until the unlock time, calculated as `startTime + duration`, has passed.

The issue is that traders can manipulate the `startTime`, allowing them to potentially unstake their tokens earlier than intended by altering the `startTime`.

src/Endpoint.sol:L230-L252

```
function stakeLogX(StakeLogXRequest memory stakeLogXRequest) internal {
    ...
    clearinghouse.stakeLogXForAccount(stakeLogXRequest.stakerContract,
stakeLogXRequest.subAccountId, amountInUint, stakeLogXRequest.duration,
stakeLogXRequest.startTime);
    ...
}
```

Recommendation

`startTime` for one stake should not be less than `current.timestamp`.

Status

This issue has been resolved by the team via off-chain check.

11. stLogX may be cleared in liquidation process

Severity: Medium	Category: Business Logic
Target: - src/Clearinghouse.sol	

Description

When a trading account is deemed unhealthy, the sequencer will initiate the liquidation of the account's positions. During this process, the liquidated account's spot product balance, including stLogX, may be entirely cleared.

The issue is that the liquidated account can still claim rewards even after their stLogX spot balance has been cleared. Additionally, the protocol team is unable to unstake these stLogX tokens into LogX.

src/Clearinghouse.sol:L497-L522

```
function _settleLiqPnlUsingSpots(
    IEndpoint.LiquidateSubaccount calldata txn,
    ISpotEngine spotEngine,
    int128 totalPnl
) private returns (int128) {
    ...
    // Check if the user has enough balance to pay the PnL
    if (balance.amount.mul(txn.spotPricesX18[i]) < -totalPnl) {
        // If not, then update the balance to 0
        spotEngine.updateBalance(spotProductIds[i], liquidatee, -balance.amount);
        totalPnl += balance.amount.mul(txn.spotPricesX18[i]);
        spotEngine.updateAvailableSettle(spotProductIds[i], balance.amount);
    }
    ...
}
```

Recommendation

Consider to force unstaking liquidatees' staking and convert stLogX to LogX. This will depend on the staking contract's implementation.

Status

This issue has been resolved by the team via off-chain check.

12. Lack of bridgeOutContract verification in withdrawLogX

Severity: Medium	Category: Centralization
Target: - src/Clearinghouse.sol	

Description

Traders can withdraw LogX by `withdrawLogX` API. The LogX hyper_erc20 will help bridge LogX to the expected destination.

The problem is that the traders can manipulate the `bridgeOutContract` parameter to transfer some other collaterals from the clearinghouse contract.

src/Clearinghouse.sol:L215-L234

```
function withdrawLogX(WithdrawLogX memory withdrawRequest) external onlyEndpoint {
    ...
    bytes32 receiver = bytes32(uint256(uint160(withdrawRequest.receiver)));
    //update user balance
    int128 amountInInt = -(amountAfterFees);
    // Decrease the sub Account's balance
    spotEngine.updateBalance(LOGX_PRODUCT_ID, withdrawRequest.subAccountId,
amountInInt);
    //call tokenrouter contract
    IHyperlane(withdrawRequest.bridgeOutContract).transferRemote{value:
uint128(amountAfterFees)}(uint32(withdrawRequest.destinationChainId),receiver,
uint128(amountAfterFees));
    ...
}
```

Recommendation

Check whether the off-chain backend has verified this input parameter. Suggest to add the related input parameter on-chain.

Status

This issue has been resolved by the team with commit 992c7ae.

13. Lack of stakerContract verification in claimRewards

Severity: Medium	Category: Centralization
Target: - src/Endpoint.sol	

Description

In LogX, traders can stake their LogX tokens to earn rewards. They can then claim these rewards through the `ClaimRewards` API.

The issue is that stakers can manipulate the `stakerContract` parameter in the `ClaimRewards` API. Malicious users can exploit this by using a fraudulent `stakerContract` to receive disproportionately large reward amounts, thereby gaining undue profit.

src/Endpoint.sol:L272-L281

```
function claimRewards(ClaimRewards memory claimRewardsRequest ) internal {
    address masterWallet = accountInfo[claimRewardsRequest.subAccountId].userAddress;
    //signature verification
    uint256 rewards = clearinghouse.claimRewards(claimRewardsRequest.stakerContract,
claimRewardsRequest.subAccountId);
    int128 rewardsInInt = int128(int(rewards));
    spotEngine.updateBalance(claimRewardsRequest.productId,
claimRewardsRequest.subAccountId,rewardsInInt );
    emit ClaimStakeRewards(claimRewardsRequest.subAccountId, masterWallet, rewards);
}
```

src/Clearinghouse.sol:L152-L155

```
function claimRewards(address stakerContract, bytes32 subAccount) external onlyEndpoint
returns(uint256 amount) {
    amount = IStaker(stakerContract).claimTokensForAccount(subAccount, address(this));
}
```

Recommendation

Check whether the off-chain backend has verified this input parameter. Suggest to add the related input parameter on-chain.

Status

This issue has been resolved by the team via off-chain check.

14. Lack of productId verification in stakeLogX

Severity: Medium	Category: Centralization
Target: - src/Endpoint.sol	

Description

In LogX, traders can stake their LogX tokens to earn rewards. Upon staking, the trader's LogX spot balance decreases, and their stLogX spot balance increases accordingly. However, a problem arises because stakers can manipulate the `productId` parameter in the `stakeLogX` function.

This manipulation allows traders to increase the balance of a different spot token instead of stLogX, thereby obtaining unauthorized profits.

src/Endpoint.sol:L230-L252

```
function stakeLogX(StakeLogXRequest memory stakeLogXRequest) internal {
    ...
    uint256 amountInUint = uint128(stakeLogXRequest.tokenAmount);
    clearinghouse.stakeLogXForAccount(stakeLogXRequest.stakerContract,
stakeLogXRequest.subAccountId, amountInUint, stakeLogXRequest.duration,
stakeLogXRequest.startTime);
    int128 amountInInt = -(int128(stakeLogXRequest.tokenAmount));
    spotEngine.updateBalance(LOGX_PRODUCT_ID, stakeLogXRequest.subAccountId,
amountInInt);
    spotEngine.updateBalance(LOGX_PRODUCT_ID, STAKER_ACCOUNT,
stakeLogXRequest.tokenAmount);
    spotEngine.updateBalance(stakeLogXRequest.productId, stakeLogXRequest.subAccountId,
stakeLogXRequest.tokenAmount);
    emit StakeLogX(stakeLogXRequest.subAccountId, masterWallet, amountInUint);
}
```

Recommendation

Check whether the off-chain backend has verified this input parameter. Suggest to add the related input parameter on-chain.

Status

This issue has been resolved by the team with commit 992c7ae.

15. Lack of tokenAmount verification in claimLogX

Severity: Medium	Category: Centralization
Target: - src/Endpoint.sol	

Description

In LogX, sequencers deposit LogX into the Endpoint contract for airdrop purposes. Users can then claim their airdropped LogX by using the `claimLogX` function.

The issue is that we do not perform on-chain validation of the `tokenAmount`. As a result, users can exploit this by manipulating the `tokenAmount` to claim more LogX than they are entitled to.

src/Endpoint.sol:L306-L316

```
function claimLogX(ClaimLogX memory depositRequest) internal {
    uint128 tokenAmount = uint128(depositRequest.tokenAmount);
    (bool status, ) = payable(address(clearinghouse)).call{value: tokenAmount}("");
    require(status, "EP: LogX transfer Failed");
    address masterWallet = accountInfo[depositRequest.subAccountId].userAddress;
    ISpotEngine(spotEngine).updateBalance(LOGX_PRODUCT_ID, depositRequest.subAccountId,
depositRequest.tokenAmount);
    emit ClaimedLogX(depositRequest.subAccountId, masterWallet,
depositRequest.tokenAmount);
}
```

Recommendation

Check whether the off-chain backend has verified this input parameter. Suggest to add the related input parameter check on-chain.

Status

This issue has been resolved by the team via off-chain check.

16. Lack of productId verification in claimRewards

Severity: Medium	Category: Centralization
Target: - src/Endpoint.sol	

Description

In LogX, users can claim their staking rewards by `claimReward` API.

The issue is that the `productId` is not validated on-chain. This oversight allows users to claim LogX rewards and manipulate the balance of a different collateral, resulting in unintended profits.

src/Endpoint.sol:L272-L281

```
function claimRewards(ClaimRewards memory claimRewardsRequest ) internal {
    address masterWallet = accountInfo[claimRewardsRequest.subAccountId].userAddress;
    //signature verification
    uint256 rewards = clearinghouse.claimRewards(claimRewardsRequest.stakerContract,
claimRewardsRequest.subAccountId);
    int128 rewardsInInt = int128(int(rewards));
    spotEngine.updateBalance(claimRewardsRequest.productId,
claimRewardsRequest.subAccountId,rewardsInInt );
    emit ClaimStakeRewards(claimRewardsRequest.subAccountId, masterWallet, rewards);
}
```

Recommendation

Check whether the off-chain backend has verified this input parameter. There is not any record about users' airdrop amount on-chain. This check has to be done by backend sequencers.

Status

This issue has been resolved by the team with commit 992c7ae.

17. Lack of tokenAmount verification in stakeLogX

Severity: Medium	Category: Centralization
Target: - src/Endpoint.sol	

Description

Traders can stake LogX to earn rewards using the `StakeLogRequest` API.

However, the issue lies in the inadequate on-chain validation of the `tokenAmount`. If a negative value is provided for `tokenAmount`, casting it from `int128` to `uint128` will result in an overflow, leading to an excessively large value for `amountInUint`. This vulnerability allows users to potentially receive disproportionately large staking rewards.

src/Endpoint.sol:L216-L231

```
function stakeLogX(StakeLogXRequest memory stakeLogXRequest) internal {
    int128 userBalance = spotEngine.getBalance(LOGX_PRODUCT_ID,
stakeLogXRequest.subAccountId).amount;
    require(userBalance >= stakeLogXRequest.tokenAmount, "User doesnt have sufficient
balance");
    address masterWallet = accountInfo[stakeLogXRequest.subAccountId].userAddress;
    uint256 amountInUint = uint128(stakeLogXRequest.tokenAmount);
    clearinghouse.stakeLogXForAccount(stakeLogXRequest.stakerContract,
stakeLogXRequest.subAccountId, amountInUint, stakeLogXRequest.duration,
stakeLogXRequest.startTime);
    ...
}
```

Recommendation

Check whether the off-chain backend has verified this input parameter. Suggest to add the related input parameter check on-chain.

Status

This issue has been resolved by the team with commit ae10cbb.

18. Missing minSize order size check

Severity: Low	Category: Business Logic
Target: - src/PerpEngine.sol - src/SpotEngine.sol - src/OffChainExchange.sol	

Description

The variable `minSize` in OffChainExchange is used to limit the minimum size of one trading size.

The problem is that we miss order's `minSize` check when we match the order.

Recommendation

Check if the position size matches the `minSize` order size limitation in the process of matching orders.

Status

This issue has been resolved by the team with commit 4f9a5ab.

19. Some orders will never be fully filled.

Severity: Low	Category: Data Validation
Target: - src/OffChainExchange.sol	

Description

During the order execution process, the trade size is restricted to integer multiples of `sizeIncrement`. Therefore, orders with `order.amount` that are not integer multiples of `sizeIncrement` will never be fully filled. The portion that is not a multiple of `sizeIncrement` can still proceed with order matching, preventing the trade from reverting but remaining unfilled. If the `sequencer` relies on whether a trade reverts to match and execute orders, this could increase the probability of the `sequencer` executing invalid order matches.

src/OffChainExchange.sol:L269-L284

```
function _matchMakerOrder(
    CallState memory callState,
    MarketInfo memory market,
    IEndpoint.Order memory taker,
    IEndpoint.Order memory maker,
    OrdersInfo memory ordersInfo,
    address makerLinkedSigner
) internal returns (int128 takerAmountDelta, int128 takerQuoteDelta, int128 makerFee) {
    ...
    takerAmountDelta -= takerAmountDelta % market.sizeIncrement;
    int128 makerQuoteDelta = takerAmountDelta.mul(maker.priceX18);
    takerQuoteDelta = -makerQuoteDelta;
    ...
    taker.amount -= takerAmountDelta;
    maker.amount += takerAmountDelta;
    ...
}
```

Attach Scenario

1. The `sizeIncrement` for the market with `produceId` 1 is 1e18.
2. The taker order size is 110e18, and the maker order size is -(100e18 + 9e17).
3. The two orders are matched. Due to the `sizeIncrement` restriction, the maker order can only be filled up to 100e18, leaving 9e17 that will never be filled.
4. The two orders are matched again. The transaction does not revert (which it should revert if the order was fully filled), but the positions of the taker and maker do not change.

Recommendation

In the `_validateOrder` function, check if the unfilled amount of the order is less than `sizeIncrement`. If so, return false to prevent invalid transactions from proceeding. Alternatively, ensure that the order size is a multiple of `sizeIncrement`.

Status

This issue has been resolved by the team with commit 4f9a5ab.

2.3 Informational Findings

20. Missing gap

Severity: Informational	Category: Code Quality
Target: - src/BaseEngine.sol	

Description

The `BaseEngine` contract is inherited by both the `PerpEngine` and `SpotEngine` contracts. However, BaseEngine does not implement storage gaps. Without these gaps, adding new storage variables to the `BaseEngine` contract could potentially overwrite the initial storage layout of the child contracts, leading to conflicts and unintended behavior.

Recommendation

Add storage `gaps` to the BaseEngine contract.

Status

This issue has been resolved by the team with commit 992c7ae.

21. The error message lacks specificity

Severity: Informational	Category: Code Quality
Target: - src/SpotEngine.sol	

Description

The `manualAssert` function verifies that the `totalDeposits` for each product in the market aligns with the expected values. If any discrepancies are detected, the transaction will revert. However, the error handling does not specify which product's `totalDeposits` is mismatched; instead, it returns a generic error message.

src/SpotEngine.sol:L147-L158

```
function manualAssert(
    int128[] calldata totalDeposits
) external view {
    for (uint128 i = 0; i < totalDeposits.length; ++i) {
        uint32 productId = productIds[i];
        TokenInfo memory _tokenInfo = tokenInfo[productId];
        require(
            _tokenInfo.totalDeposits == totalDeposits[i],
            ERR_DSYNC
        );
    }
}
```

Recommendation

Consider returning an error message with the `produceId` of the product whose `totalDeposits` does not match.

Status

This issue has been acknowledged by the team.

22. Possible hash collision in getDigest

Severity: Informational	Category: Data Validation

Target:
- src/OffChainExchange.sol

Description

Traders may submit two orders with identical price, amount, expiration, and other parameters, resulting in the same digest for both orders. The contract cannot differentiate between these duplicate orders. Consequently, after the first order is fully filled, the second order cannot be executed.

src/OffChainExchange:L148-L171

```
function getDigest(uint32 productId, IEndpoint.Order memory order)
    public
    view
    returns (bytes32)
{
    ...
    bytes32 structHash = keccak256(
        abi.encode(
            keccak256(bytes(structType)),
            order.subAccountId,
            order.priceX18,
            order.amount,
            order.expiration,
            order.isReduce,
            order.sessionKey,
            order.chainId,
            productId
        )
    );
    return SignatureVerifier.getDigest(structHash, LOGX_CHAIN_ID,
address(virtualBookContract[productId]));
}
```

Recommendation

Improve the documentation to state that identical orders can only be executed once.

Status

This issue has been resolved by the team and the team will add this into the document.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 3b3bbde:

File	SHA-1 hash
OffChainExchange.sol	b3dba0488bef3e93ad172fdc4ed2c3e42c0d8969
Clearinghouse.sol	8790143579824f8ef7aa2c0e3bc5e9af387876b8
Endpoint.sol	4ce8137db5c23ac7bc2269c7ae85b842ad6671bd
PerpEngineState.sol	7d71b0f27d67ffb27eaa6a45f83ae0c86ea0ed85
PerpEngine.sol	42e38558f1c739f4f0257afe8875423c4c7778f0
SpotEngine.sol	5e49f9e1c543e7255a2df09c716a72cd3f0f271d
SpotEngineState.sol	df56e5d0e769db1ced78cc35cec05c13b2a72c74
BaseEngine.sol	de945850a5ac5c7b4c16f3d1a04800adcffc5c45
EndpointGated.sol	8a55bdc7427bb817bb5b5115e86194d611990467
ClearinghouseStorage.sol	8790143579824f8ef7aa2c0e3bc5e9af387876b8
Version.sol	1f33cd41312af5b92c73203096e97f886e85a105
Errors.sol	4cf3c60cb2ef3293e85cd236910a1ebb6355f54c
Constants.sol	623575efde00bfb5c01e14ee6239577488fed408
Logger.sol	2a6b0252d2e08deff33095d869688850721c8b9d
MathSD21x18.sol	efb90519e4efc651dc7676fa7d39bc700c1d7af6
ERC20Helper.sol	1da4fea3e65ba9426b661f0a4f5923fcd63ae2c2
MathHelper.sol	b2c32dfb941acdfc7b6ac9540317c6581b3f0551
RiskHelper.sol	e2dd5fde3f1acd82c0ed013c922d2159f004698e
SignatureVerifier.sol	9adb68a08825189e637e1a9237e5c812c76f59e7