CODE SECURITY ASSESSMENT

APOLLOX
Overview

Project Summary

- Name: ApolloX
- Version: commit c48d047
- Platform: BNB Smart Chain
- Language: Solidity
- Repository: https://github.com/apollox-finance/apollox-perp-contracts
- Audit Scope: See Appendix - 1

Project Dashboard

Application Summary

<table>
<thead>
<tr>
<th>Name</th>
<th>ApolloX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>v2</td>
</tr>
<tr>
<td>Type</td>
<td>Solidity</td>
</tr>
<tr>
<td>Dates</td>
<td>Apr 20 2023</td>
</tr>
<tr>
<td>Logs</td>
<td>Apr 18 2023; Apr 20 2023</td>
</tr>
</tbody>
</table>

Vulnerability Summary

<table>
<thead>
<tr>
<th>Issue Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total High-Severity issues</td>
<td>1</td>
</tr>
<tr>
<td>Total Medium-Severity issues</td>
<td>4</td>
</tr>
<tr>
<td>Total Low-Severity issues</td>
<td>8</td>
</tr>
<tr>
<td>Total informational issues</td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td>17</td>
</tr>
</tbody>
</table>

Contact

E-mail: support@salusec.io
## Risk Level Description

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>High Risk</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Medium Risk</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Low Risk</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Informational</strong></td>
<td>The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.</td>
</tr>
</tbody>
</table>
Content

Introduction 4
  1.1 About SALUS 4
  1.2 Audit Breakdown 4
  1.3 Disclaimer 4

Findings 5
  2.1 Summary of Findings 5
  2.2 Notable Findings 6
    1. Flawed logic for updating lpAveragePrice 6
    2. Using the longAccFundingFeePerShare after it has been deleted prevents the trader from closing the position 7
    3. Data is written to memory instead of storage 9
    4. Incorrect calculation in checkTP() 10
    5. Centralization risk 11
    6. Improper checks for token existence 13
    7. Fee on transfer token can lead to incorrect reserves balances for the reward pool 14
    8. Use of uninitialized variable in the return statement 15
    9. Oracle return values are not used properly 16
    10. Imprecise bounds 18
    11. Potential divided by zero error 20
    12. removePair() would not clean the pair’s leverageMargins mapping 22
    13. Missing events for important parameter change 23

  2.3 Informational Findings 24
    14. SafeMath library not needed since Solidity 0.8.0 24
    15. Lack of validation for the price feed in addChainlinkPriceFeed() 25
    16. Spelling error 26
    17. Inconsistent naming convention 27

Appendix 28
  Appendix 1 - Files in Scope 28
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

<table>
<thead>
<tr>
<th>ID</th>
<th>Title</th>
<th>Severity</th>
<th>Category</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Flawed logic for updating lpAveragePrice</td>
<td>High</td>
<td>Business logic</td>
<td>Resolved</td>
</tr>
<tr>
<td>2</td>
<td>Using the longAccFundingFeePerShare after it has been deleted prevents the trader from closing the position</td>
<td>Medium</td>
<td>Business logic</td>
<td>Resolved</td>
</tr>
<tr>
<td>3</td>
<td>Data is written to memory instead of storage</td>
<td>Medium</td>
<td>Business logic</td>
<td>Resolved</td>
</tr>
<tr>
<td>4</td>
<td>Incorrect calculation in checkTP()</td>
<td>Medium</td>
<td>Business logic</td>
<td>Resolved</td>
</tr>
<tr>
<td>5</td>
<td>Centralization risk</td>
<td>Medium</td>
<td>Centralization</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>6</td>
<td>Improper checks for token existence</td>
<td>Low</td>
<td>Validation</td>
<td>Resolved</td>
</tr>
<tr>
<td>7</td>
<td>Fee on transfer token can lead to incorrect reserves balances for the reward pool</td>
<td>Low</td>
<td>Business logic</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>8</td>
<td>Use of uninitialized variable in the return statement</td>
<td>Low</td>
<td>Business logic</td>
<td>Resolved</td>
</tr>
<tr>
<td>9</td>
<td>Oracle return values are not used properly</td>
<td>Low</td>
<td>Business logic</td>
<td>Resolved</td>
</tr>
<tr>
<td>10</td>
<td>Imprecise bounds</td>
<td>Low</td>
<td>Validation</td>
<td>Resolved</td>
</tr>
<tr>
<td>11</td>
<td>Potential divided by zero error</td>
<td>Low</td>
<td>Arithmetic issue</td>
<td>Resolved</td>
</tr>
<tr>
<td>12</td>
<td>removePair() would not clean the pair’s leverageMargins mapping</td>
<td>Low</td>
<td>Business logic</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>13</td>
<td>Missing events for important parameter change</td>
<td>Low</td>
<td>Logging</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>14</td>
<td>SafeMath library not needed since Solidity 0.8.0</td>
<td>Informational</td>
<td>Redundancy</td>
<td>Resolved</td>
</tr>
<tr>
<td>15</td>
<td>Lack of validation for the price feed in addChainlinkPriceFeed()</td>
<td>Informational</td>
<td>Validation</td>
<td>Resolved</td>
</tr>
<tr>
<td>16</td>
<td>Spelling error</td>
<td>Informational</td>
<td>Code quality</td>
<td>Resolved</td>
</tr>
<tr>
<td>17</td>
<td>Inconsistent naming convention</td>
<td>Informational</td>
<td>Code quality</td>
<td>Resolved</td>
</tr>
</tbody>
</table>
2.2 Notable Findings

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

<table>
<thead>
<tr>
<th>1. Flawed logic for updating lpAveragePrice</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Severity:</strong> High</td>
</tr>
<tr>
<td><strong>Category:</strong> Business logic</td>
</tr>
<tr>
<td><strong>Target:</strong></td>
</tr>
<tr>
<td>- contracts/diamond/facets/TradingCoreFacet.sol</td>
</tr>
</tbody>
</table>

**Description**

The variable pairPositionInfos in TradingCoreStorage stores the position information of trading pairs. The pairPositionInfos variable contains the lpAveragePrice variable, which records the average price of LP’s positions if the LP were to open or close a position in the opposite direction when a trader opens or closes a position.

However, the logic behind lpAveragePrice’s updating is flawed, it does not consider the entry price when closing a position. For example, when a trader closes a long position with entryPrice P, the lpAveragePrice should be updated as if the LP closed a short position with entryPrice P. But in the code, the lpAveragePrice is updated as if the LP closed a short position with entryPrice of last lpAveragePrice (i.e. only the position quantity is changed, the lpAveragePrice remains the same). Therefore if P != last lpAveragePrice, the lpAveragePrice is updated incorrectly.

Consider the following scenario:

1. Assume the market for BTC/USD pair is empty, and the openFee, closeFee, and fundingRate are all set to zero.
2. Alice opens a long position with an entry price of $22,000 and a quantity of 1e10. This updates the lpAveragePrice to $22,000.
3. Bob then opens a long position with an entry price of $23,000 and a quantity of 1e10. This updates the lpAveragePrice to the average of both positions, $22,500.
4. Alice closes the position and Bob keeps the position in the market. The lpAveragePrice remains $22,500. But actually the lpAveragePrice should be $23,000 so that the unrealized profit/loss for LP equals the unrealized profit/loss for Bob.

The lpAveragePrice variable is used to calculate the LP’s unrealized profit and loss (lpUnrealizedPnlUsd). This, in turn, is used to calculate the ALP price. Therefore, when lpAveragePrice is incorrectly updated, the ALP price will be miscalculated.

**Recommendation**

We recommend taking the entry price of the position into account when updating the lpAveragePrice.

**Status**

The team has resolved this issue in commit 6ae69e1.
2. Using the longAccFundingFeePerShare after it has been deleted prevents the trader from closing the position

Severity: Medium  Category: Business logic

Target:  
- contracts/diamond/facets/TradingCoreFacet.sol

Description

contracts/diamond/facets/TradingCloseFacet.sol:L34-L55

function _closeTrade(
    LibTrading.TradingStorage storage ts, OpenTrade storage ot,
    bytes32 tradeHash, uint256 marketPrice, uint256 closePrice
) private returns (IOrderAndTradeHistory.CloseInfo memory) {
    int256 longAccFundingFeePerShare =
        ITradingCore(address(this)).updatePairPositionInfo(ot.pairBase, closePrice, marketPrice, ot.qty, ot.isLong, false);
    IVault.MarginToken memory mt = IVault(address(this)).getTokenForTrading(ot.tokenIn);
    int256 fundingFee = LibTrading.calcFundingFee(ot, mt, marketPrice, longAccFundingFeePerShare);
    uint256 closeNotionalUsd = closePrice * ot.qty;
    int256 pnl =
        ot.isLong
            ? int256(closeNotionalUsd) - int256(uint256(ot.entryPrice * ot.qty))
            : int256(uint256(ot.entryPrice * ot.qty)) - int256(closeNotionalUsd)
        * int256(10 ** mt.decimals) / int256(1e10 * mt.price);
    uint16 closeFeeP =
        IPairsManager(address(this)).getPairFeeConfig(ot.pairBase).closeFeeP;
    uint256 closeFee =
        closeNotionalUsd * closeFeeP * (10 ** mt.decimals) / (1e4 * 1e10 * mt.price);
    _settleForCloseTrade(ts, ot, tradeHash, pnl, fundingFee, closeFee);

    return IOrderAndTradeHistory.CloseInfo(uint64(closePrice), int96(fundingFee), uint96(closeFee), int96(pnl));
}

When closing a position, the longAccFundingFeePerShare value returned from updatePairPositionInfo() is used to calculate and settle the funding fee in subsequent code.

contracts/diamond/facets/TradingCoreFacet.sol:L90-L109

function updatePairPositionInfo(
    address pairBase, uint userPrice, uint marketPrice, uint qty, bool isLong, bool isOpen
) external onlySelf override returns (int256 longAccFundingFeePerShare){
    LibTradingCore.TradingCoreStorage storage tcs = LibTradingCore.tradingCoreStorage();
    PairPositionInfo storage ppi = tcs.pairPositionInfos[pairBase];
    if (ppi.longQty > 0 || ppi.shortQty > 0) {
        uint256 lpReceiveFundingFeeUsd = _updateFundingFee(ppi, pairBase, marketPrice);
        if (lpReceiveFundingFeeUsd > 0) {
            ITradingPortal(address(this)).settleLpFundingFee(lpReceiveFundingFeeUsd);
        }
    }
}


ppi.lastFundingFeeBlock = block.number;
}
_updatePairQtyAndAvgPrice(tcs, ppi, pairBase, qty, userPrice, isOpen, isLong);
emit UpdatePairPositionInfo(
    pairBase, ppi.lastFundingFeeBlock, ppi.longQty, ppi.shortQty, ppi.longAccFundingFeePerShare, ppi.lpAveragePrice
);
return ppi.longAccFundingFeePerShare;
}

contracts/diamond/facets/TradingCoreFacet.sol:L153-L243

function _updatePairQtyAndAvgPrice(
    LibTradingCore.TradingCoreStorage storage tcs, ITradingCore.PairPositionInfo storage ppi, address pairBase, uint256 qty, uint256 userPrice, bool isOpen, bool isLong
) private {
    if (isOpen) {
    ...
    } else {
        if (isLong) {
            ...
        } else {
            ...
        }
    }
    if (ppi.longQty == 0 && ppi.shortQty == 0) {
        address[] storage pairs = tcs.hasPositionPairs;
        uint lastIndex = pairs.length - 1;
        uint removeIndex = ppi.pairIndex;
        if (lastIndex != removeIndex) {
            address lastPair = pairs[lastIndex];
            pairs[removeIndex] = lastPair;
            tcs.pairPositionInfos[lastPair].pairIndex = uint16(removeIndex);
        }
        delete tcs.pairPositionInfos[pairBase];
    }
}

However, when closing the last position in the market for a pair, the _updatePairQtyAndAvgPrice() function deletes the position info for the pair before the the longAccFundingFeePerShare value in the position info is returned. As a result, the longAccFundingFeePerShare returned from the updatePairPositionInfo() will be zero in this case and the position may not be able to be closed due to incorrect funding fee.

**Recommendation**

We recommend setting the return value longAccFundingFeePerShare to ppi.longAccFundingFeePerShare before executing _updatePairQtyAndAvgPrice() in the updatePairPositionInfo() function.

**Status**

The team has resolved this issue in commit [152f3fb](#).
3. Data is written to memory instead of storage

<table>
<thead>
<tr>
<th>Severity: Medium</th>
<th>Category: Business logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target:</td>
<td>contracts/diamond/libraries/LibBrokerManager.sol</td>
</tr>
</tbody>
</table>

**Description**

contracts/diamond/libraries/LibBrokerManager.sol:L104-L110

```solidity
function updateBrokerCommissionP(uint24 id, uint16 commissionP) internal {
    BrokerManagerStorage storage bms = brokerManagerStorage();
    Broker memory b = _checkBrokerExist(bms, id);
    uint16 oldCommissionP = b.commissionP;
    b.commissionP = commissionP;
    emit UpdateBrokerCommissionP(id, oldCommissionP, commissionP);
}
```

The commissionP is written to b.commissionP, which is a variable stored in memory, not in storage. That is, the updateBrokerCommissionP() does not update the commissionP storage variable.

contracts/diamond/libraries/LibBrokerManager.sol:L112-L118

```solidity
function updateBrokerReceiver(uint24 id, address receiver) internal {
    BrokerManagerStorage storage bms = brokerManagerStorage();
    Broker memory b = _checkBrokerExist(bms, id);
    address oldReceiver = b.receiver;
    b.receiver = receiver;
    emit UpdateBrokerReceiver(id, oldReceiver, receiver);
}
```

contracts/diamond/libraries/LibBrokerManager.sol:L120-L126

```solidity
function updateBrokerName(uint24 id, string calldata name) internal {
    BrokerManagerStorage storage bms = brokerManagerStorage();
    Broker memory b = _checkBrokerExist(bms, id);
    string memory oldName = b.name;
    b.name = name;
    emit UpdateBrokerName(id, oldName, name);
}
```

contracts/diamond/libraries/LibBrokerManager.sol:L128-134

```solidity
function updateBrokerUrl(uint24 id, string calldata url) internal {
    BrokerManagerStorage storage bms = brokerManagerStorage();
    Broker memory b = _checkBrokerExist(bms, id);
    string memory oldUrl = b.url;
    b.url = url;
    emit UpdateBrokerUrl(id, oldUrl, url);
}
```

Likewise, the updateBrokerReceiver(), updateBrokerName(), and updateBrokerUrl() functions do not update the parameters in storage.

**Recommendation**

Consider writing the parameters to storage, rather than to memory.

**Status**

The team has resolved this issue in commit c05a022.
4. Incorrect calculation in checkTP()

Severity: Medium  
Category: Business logic

Target:  
- contracts/diamond/facets/TradingCheckerFacet.sol

Description

countacts/diamond/facets/TradingCheckerFacet.sol:L16-L26

```solidity
function checkTp(
    bool isLong, uint takeProfit, uint entryPrice, uint leverage_10000, uint maxTakeProfitP
) public pure returns (bool) {
    if (isLong) {
        // The takeProfit price must be set and the percentage of profit must not exceed
        // the maximum allowed
        return takeProfit > entryPrice && (takeProfit - entryPrice) * leverage_10000 <=
            maxTakeProfitP * entryPrice;
    } else {
        // The takeProfit price must be set and the percentage of profit must not exceed
        // the maximum allowed
        return takeProfit > 0 && takeProfit < entryPrice && (entryPrice - takeProfit) *
            leverage_10000 <= maxTakeProfitP * takeProfit;
    }
}
```

When calculating the profit ratio, the profit should be compared to its entry notional value. Thus, the highlighted `takeProfit` should be replaced with `entryPrice`.

Recommendation

Consider change the line
```solidity
return takeProfit > 0 && takeProfit < entryPrice && (entryPrice - takeProfit) *
    leverage_10000 <= maxTakeProfitP * takeProfit;
```
to
```solidity
return takeProfit > 0 && takeProfit < entryPrice && (entryPrice - takeProfit) *
    leverage_10000 <= maxTakeProfitP * entryPrice;
```

Status

The team has resolved this issue in commit f35baa7.
# 5. Centralization risk

<table>
<thead>
<tr>
<th>Severity: Medium</th>
<th>Category: Centralization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target:</td>
<td>all</td>
</tr>
</tbody>
</table>

## Description

Throughout the ApolloX project, there are several privileged roles.

The **ADMIN_ROLE** can:
- Pause and resume the project
- Set DAO repurchase address
- Set DAO share percent
- Set price gap and max delay for price request callback
- Set security margin percent for the vault
- Set broker configs
- Set trading configs
- Set cooling duration for the ALP token

The **DEPLOYER_ROLE** can:
- Use diamondCut() to update diamond facets (i.e. upgrade functions)

The **TOKEN_OPERATOR_ROLE** can:
- Configure tokens in the vault

The **STAKE_OPERATOR_ROLE** can:
- Update the number of APX reward per block

The **PRICE_FEED_OPERATOR_ROLE** can:
- Configure price feed for a token

The **PAIR_OPERATOR_ROLE** can:
- Add or remove trading pairs
- Updating the pair status
- Configure fee data for a pair
- Configure slippage data for a pair
- Configure funding fee for a pair
- Configure leverage and maxOI for a pair

The **KEEPER_ROLE** is responsible for:
- Executing the limit orders
- Executing the take profit orders
- Executing the stop loss orders
- Executing the liquidations
The PRICE_FEEDER_ROLE is responsible for:
- Responding to the price request

And most importantly, the DEFAULT_ADMIN_ROLE can:
- Add or remove an address to the previously mentioned roles

If an attacker were to gain access to the private keys associated with these roles, they could cause significant damage to the project.

Recommendation

We recommend transferring the privileged roles to multi-sig accounts.

Status

This issue has been acknowledged by the team.
6. Improper checks for token existence

<table>
<thead>
<tr>
<th>Severity: Low</th>
<th>Category: Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Target:</strong></td>
<td></td>
</tr>
<tr>
<td>- contracts/diamond/libraries/LibVault.sol</td>
<td></td>
</tr>
<tr>
<td>- contracts/diamond/libraries/LibAlpManager.sol</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

contracts/diamond/libraries/LibVault.sol:L83  
contracts/diamond/libraries/LibVault.sol:L104  
contracts/diamond/libraries/LibVault.sol:L123  
contracts/diamond/libraries/LibVault.sol:L136

```solidity
require(at.weight > 0, "LibVault: Token does not exist");
```

contracts/diamond/libraries/LibAlpManager.sol:L71  
contracts/diamond/libraries/LibAlpManager.sol:L107

```solidity
require(at.weight > 0, "LibAlpManager: Token does not exist");
```

In the above lines, the condition `at.weight > 0` is used to check if the token exists. However, the weight for an existing token can be set to zero by the `TOKEN_OPERATOR_ROLE`.

**Recommendation**

Consider checking if the token exists by using the condition `at.tokenAddress != address(0)`.

**Status**

The team has resolved this issue in commit [8af36b6](#) and [07103b4](#).
7. Fee on transfer token can lead to incorrect reserves balances for the reward pool

<table>
<thead>
<tr>
<th>Severity: Low</th>
<th>Category: Business logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target:</td>
<td>contracts/diamond/libraries/LibApxReward.sol</td>
</tr>
</tbody>
</table>

**Description**

contracts/diamond/libraries/LibApxReward.sol:L145-L151

```solidity
function addReserves(uint256 amount) internal {
    ApxRewardStorage storage ars = apxRewardStorage();
    IApxReward.ApxPoolInfo storage pool = ars.poolInfo;
    ars.rewardToken.transferFrom(msg.sender, address(this), amount);
    pool.reserves += amount;
    emit AddReserves(msg.sender, amount);
}
```

The `ars.rewardToken` is expected to be set to the APX token, a fee-on-transfer token. Thus, the amount of the reward token received from the `transferFrom()` call may be less than the transfer amount. However, the pool's reserves are increased by the transfer amount, not the received amount.

**Recommendation**

Consider checking the balance before and after the transfer, and using the difference as the amount of tokens received, then adding the received amount to `pool.reserves`.

**Status**

The team has acknowledged this issue. The code remains unchanged because the ApolloX contract is whitelisted by ALP, and transferring ALP tokens to the ApolloX contract incurs no fees. Therefore, the amount received for the reward pool is equal to the amount sent by the user.
8. Use of uninitialized variable in the return statement

<table>
<thead>
<tr>
<th>Severity: Low</th>
<th>Category: Business logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target:</td>
<td>contracts/diamond/facets/TradingCheckerFacet.sol</td>
</tr>
</tbody>
</table>

**Description**

The above path in the marketTradeCallbackCheck() function returns a tuple. Its second element, entryPrice, is an uninitialized variable and defaults to zero.

Instead of using `entryPrice`, the above code should use `tuple.entryPrice`.

**Recommendation**

Consider changing `entryPrice` to `tuple.entryPrice`.

**Status**

The team has resolved this issue in commit c5a429a.
9. Oracle return values are not used properly

Severity: Low Category: Business logic

Target:
- contracts/diamond/libraries/LibChainlinkPrice.sol
- contracts/diamond/libraries/LibPriceFacade.sol

Description

The oracle.latestRoundData() function, as documented, returns the following variables:
- uint80 roundId
- int256 answer
- uint256 startedAt
- uint256 updatedAt
- uint80 answeredInRound

Among these, startedAt is the timestamp when the price was submitted, and updatedAt is the timestamp when the price was updated on the chain.

contracts/diamond/libraries/LibChainlinkPrice.sol:L60-L69

```solidity
function getPriceFromChainlink(address token) internal view returns (uint256 price, uint8 decimals, uint256 startedAt) {
    ChainlinkPriceStorage storage cps = chainlinkPriceStorage();
    address priceFeed = cps.priceFeeds[token].feedAddress;
    require(priceFeed != address(0), "LibChainlinkPrice: Price feed does not exist");
    AggregatorV3Interface oracle = AggregatorV3Interface(priceFeed);
    (, int256 price_, uint256 startedAt_,,) = oracle.latestRoundData();
    price = uint256(price_);
    decimals = oracle.decimals();
    return (price, decimals, startedAt_);
}
```

contracts/diamond/libraries/LibPriceFacade.sol:L164-L172

```solidity
function getPriceFromCacheOrOracle(PriceFacadeStorage storage pfs, address token) internal view returns (uint64, uint40) {
    LatestCallbackPrice memory cachePrice = pfs.callbackPrices[token];
    (uint256 price, uint8 decimals, uint256 startedAt) = LibChainlinkPrice.getPriceFromChainlink(token);
    uint40 updatedAt = cachePrice.timestamp >= startedAt ? cachePrice.timestamp :
        uint40(startedAt);
    // Take the newer price
    uint64 tokenPrice = cachePrice.timestamp >= startedAt ? cachePrice.price :
        (decimals == 8 ? uint64(price) : uint64(price * 1e8 / (10 ** decimals)));
    return (tokenPrice, updatedAt);
}
```

The startedAt timestamp returned from oracle.latestRoundData() is used in the getPriceFromChainlink() function, but it is more appropriate to use the updatedAt timestamp instead. The timestamp returned from the getPriceFromChainlink() is then used in the getPriceFromCacheOrOracle() to check the freshness of the price.
Recommendation
Consider using `updatedAt` returned from the `oracle.latestRoundData()` to replace `startedAt`.

Status
The team has resolved this issue in commit `b4cde78`. 
10. Imprecise bounds

Severity: Low  Category: Validation

Target:
- contracts/diamond/libraries/LibAlpManager.sol
- contracts/diamond/libraries/LibVault.sol
- contracts/diamond/facets/ApxRewardFacet.sol
- contracts/diamond/libraries/LibApxReward.sol

Description

There are several checks where the value boundaries are not precise.

1. **contracts/diamond/libraries/LibAlpManager.sol:L118**

   ```solidity
   require(amountOut < vs.treasury[tokenOut], "LibAlpManager: tokenOut balance is insufficient");
   ```

   The `<` should be `<=` so that all amounts in the treasury are withdrawable.

2. **contracts/diamond/libraries/LibVault.sol:L214**

   ```solidity
   require(index.into() > 0 && otherTokenAmountUsd < totalBalanceUsd, "LibVault: Insufficient funds in the treasury");
   ```

   The highlighted `<` should `<=`.

3. **contracts/diamond/facets/ApxRewardFacet.sol:L12-L19**

   ```solidity
   function initializeApxRewardFacet(address _rewardsToken, uint256 _apxPerBlock, uint256 _startBlock) external {
       require(_rewardsToken != address(0), "Invalid _rewardsToken");
       require(_apxPerBlock >= 0, "Invalid _apxPerBlock");
       require(_startBlock >= 0, "Invalid _startBlock");
       LibAccessControlEnumerable.checkRole(Constants.DEPLOYER_ROLE);
       LibApxReward.initialize(_rewardsToken, _apxPerBlock, _startBlock);
   }
   ```

   The above `>=` should be `>`. Otherwise, the checks are unnecessary, since `_apxPerBlock` and `_startBlock` are both of type `uint256` and therefore must be `>= 0`.

4. **contracts/diamond/libraries/LibApxReward.sol:L153-L158**

   ```solidity
   function updateApxPerBlock(uint256 _apxPerBlock) internal {
       ApxRewardStorage storage st = apxRewardStorage();
       require(_apxPerBlock >= 0, "apxPerBlock greater than 0");
       updatePool();
       st.poolInfo.apxPerBlock = _apxPerBlock;
   }
   ```

   The `>=` should be `>`, as indicated in the message of the require statement.

Recommendation

Consider adjusting the boundaries in the mentioned checks.
Status

The team has resolved this issue in commit d216b69, f228efd and 879774f.
11. Potential divided by zero error

Severity: Low  
Category: Arithmetic issue

Target:  
- contracts/diamond/facets/PairsManagerFacet.sol  
- contracts/diamond/libraries/LibPairsManager.sol  
- contracts/diamond/facets/TradingCoreFacet.sol

Description

contracts/diamond/facets/PairsManagerFacet.sol:L12-L22

```
function addSlippageConfig(
    string calldata name, uint16 index, SlippageType slippageType,  
    uint256 onePercentDepthAboveUsd, uint256 onePercentDepthBelowUsd, // Allowed to be 0  
    uint16 slippageLongP, uint16 slippageShortP // Allowed to be 0
) external override {
    LibAccessControlEnumerable.checkRole(Constants.PAIR_OPERATOR_ROLE);
    require(slippageLongP < 1e4 && slippageShortP < 1e4,  
        "PairsManagerFacet: Invalid parameters");
    LibPairsManager.addSlippageConfig(index, name, slippageType,  
    onePercentDepthAboveUsd, onePercentDepthBelowUsd, slippageLongP,  
    slippageShortP);
```

contracts/diamond/libraries/LibPairsManager.sol:L120-L138

```
function addSlippageConfig(
    uint16 index, string calldata name, IPairsManager.SlippageType slippageType,  
    uint256 onePercentDepthAboveUsd, uint256 onePercentDepthBelowUsd,  
    uint16 slippageLongP, uint16 slippageShortP
) internal {
    PairsManagerStorage storage pms = pairsManagerStorage();
    SlippageConfig storage config = pms.slippageConfigs[index];
    require(!config.enable, "LibPairsManager: Configuration already exists");
    config.index = index;
    config.name = name;
    config.enable = true;
    config.slippageType = slippageType;
    config.onePercentDepthAboveUsd = onePercentDepthAboveUsd;
    config.onePercentDepthBelowUsd = onePercentDepthBelowUsd;
    config.slippageLongP = slippageLongP;
    config.slippageShortP = slippageShortP;
    emit AddSlippageConfig(index, slippageType, onePercentDepthAboveUsd,  
        onePercentDepthBelowUsd, slippageLongP, slippageShortP, name);
```

The onePercentDepthAboveUsd and onePercentDepthBelowUsd can be set to zero by the PAIR_OPERATOR_ROLE for the ONE_PERCENT_DEPTH slippageType.

contracts/diamond/facets/TradingCoreFacet.sol:L29-L49

```
function slippagePrice(
    PairQty memory pairQty,  
    IPairsManager.SlippageConfig memory sc,  
    uint256 marketPrice, uint256 qty, bool isLong
```
```solidity
global pure override returns (uint256) {
    if (isLong) {
        uint slippage = sc.slippageLongP;
        if (sc.slippageType == IPairsManager.SlippageType.ONE_PERCENT_DEPTH) {
            slippage = (pairQty.longQty + qty) * marketPrice * 1e4 / sc.onePercentDepthAboveUsd;
        }
        return marketPrice * (1e4 + slippage) / 1e4;
    } else {
        uint slippage = sc.slippageShortP;
        if (sc.slippageType == IPairsManager.SlippageType.ONE_PERCENT_DEPTH) {
            slippage = (pairQty.shortQty + qty) * marketPrice * 1e4 / sc.onePercentDepthBelowUsd;
        }
        return marketPrice * (1e4 - slippage) / 1e4;
    }
}
```

If slippageType is set to ONE_PERCENT_DEPTH and `onePercentDepthAboveUsd` is set to zero, the slippagePrice() will revert due to a divided by zero error when the user tries to open a long position or close a short position.

Likewise, if slippageType is set to ONE_PERCENT_DEPTH and `onePercentDepthBelowUsd` is set to zero, the slippagePrice() will fail when the user attempts to open a short position or close a long position.

**Recommendation**

Consider preventing the `onePercentDepthAboveUsd` and `onePercentDepthBelowUsd` from being set to zero for ONE_PERCENT_DEPTH slippageType in the addSlippageConfig() function.

**Status**

The team has resolved this issue in commit `4c0b265`. 
12. removePair() would not clean the pair’s leverageMargins mapping

Severity: Low
Category: Business logic
Target:
- contracts/diamond/libraries/LibPairsManager.sol

Description

contracts/diamond/libraries/LibPairsManager.sol:L42-L65

struct Pair {
    // tier => LeverageMargin
    mapping(uint16 => LeverageMargin) leverageMargins;
    ...}

contracts/diamond/libraries/LibPairsManager.sol:L233-L272

function removePair(address base) internal {
    ...  
    delete pms.pairs[base];
    ...
    }

The leverageMargins variable is a mapping nested in the Pair struct. The delete operator in Solidity does not clean the data in a mapping (see this section). Thus, after a pair is removed using removePair(), the leverageMargins of the pair remains in the storage. The Developers should be aware of this and ensure that, if a removed pair is added back, the new leverageMargins completely overrides the old leverageMargins.

Recommendation

Consider adding a comment to the removePair() regarding this issue.

If the tiers information is of fixed length, we recommend defining the leverageMargins as a fixed-size array instead of a mapping.

Status

The team has acknowledged this issue. A code comment has been added in commit 511b04d.
**13. Missing events for important parameter change**

<table>
<thead>
<tr>
<th>Severity: Low</th>
<th>Category: Logging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target:</td>
<td>contracts/ALP.sol</td>
</tr>
</tbody>
</table>

**Description**

 contracts/ALP.sol:L48-L66

```solidity
function addFromWhiteList(address account) external onlyRole(ADMIN_ROLE) {
    require(account != address(0), "account cannot be 0 address");
    fromWhiteList[account] = true;
}

function removeFromWhiteList(address account) external onlyRole(ADMIN_ROLE) {
    require(account != address(0), "account cannot be 0 address");
    delete fromWhiteList[account];
}

function addToWhiteList(address account) external onlyRole(ADMIN_ROLE) {
    require(account != address(0), "account cannot be 0 address");
    toWhiteList[account] = true;
}

function removeToWhiteList(address account) external onlyRole(ADMIN_ROLE) {
    require(account != address(0), "account cannot be 0 address");
    delete toWhiteList[account];
}
```

Important parameter or configuration changes should trigger an event to enable tracking off-chain. However, the `addFromWhiteList()`, `removeFromWhiteList()`, `addToWhiteList()`, and `removeToWhiteList()` functions change the whitelist - an important parameter - but do not emit events.

**Recommendation**

Consider adding events to functions that change important parameters.

**Status**

This issue has been acknowledged by the team.
2.3 Informational Findings

14. SafeMath library not needed since Solidity 0.8.0

<table>
<thead>
<tr>
<th>Severity: Informational</th>
<th>Category: Redundancy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Target:</td>
<td></td>
</tr>
<tr>
<td>- contracts/diamond/libraries/LibStakeReward.sol</td>
<td></td>
</tr>
<tr>
<td>- contracts/diamond/libraries/LibApxReward.sol</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

- contracts/diamond/libraries/LibStakeReward.sol:L11
- contracts/diamond/libraries/LibApxReward.sol:L11

using SafeMath for uint;

The SafeMath library is used to check underflow and overflow for arithmetic operations. However, since Solidity version 0.8.0, arithmetic operations revert on underflow and overflow by default.

Because the LibStakeReward and LibApxReward contracts use a Solidity version no less than 0.8.0, there is no need to use the SafeMath library.

**Recommendation**

Consider not using the SafeMath library.

**Status**

The team has resolved this issue in commit 879774f.
15. Lack of validation for the price feed in addChainlinkPriceFeed()

Severity: Informational  Category: Validation

Target:
- contracts/diamond/libraries/LibPairsManager.sol
- contracts/diamond/libraries/LibChainlinkPrice.sol

Description

contracts/diamond/libraries/LibPairsManager.sol:L165-L201

```solidity
function addPair(
    IPairsManager.PairSimple memory ps,
    uint16 slippageConfigIndex, uint16 feeConfigIndex,
    LeverageMargin[] memory leverageMargins
) internal {
    ...
    require(IPriceFacade(address(this)).getPrice(ps.base) > 0, "LibPairsManager: No price feed has been configured for the pair");
    ...
}
```

The `addPair()` function validates the price feed of a pair by checking if the price received from the feed is greater than zero.

However, the `addChainlinkPriceFeed()` function sets the price feed for a pair, but does not validate the price feed.

To maintain code consistency, the `addChainlinkPriceFeed()` function should also validate the price feed by checking if the price received is greater than zero.

**Recommendation**

Consider adding a validation for the price feed in `addChainlinkPriceFeed()`.

**Status**

The team has resolved this issue in commit 6dfd354.
16. Spelling error

Severity: Informational  Category: Code quality

Target:
- contracts/diamond/libraries/LibVault.sol
- contracts/diamond/facets/VaultFacet.sol

Description

contracts/diamond/libraries/LibVault.sol:L134
contracts/diamond/facets/VaultFacet.sol:L46

function updateAsMagin(address tokenAddress, bool asMagin)

updateAsMagin should be updateAsMargin. asMagin should be asMargin.

Recommendation
Consider correcting the spelling errors.

Status
The team has resolved this issue in commit d216b69 and 5c79041.
17. Inconsistent naming convention

Severity: Informational  Category: Code quality

Target:
- contracts/utils/Constants.sol

Description

contracts/utils/Constants.sol:L13-L28

// 0xa49807205ce4d355092ef5a8a18f56e8913cf4a201fbeb287825b095693c21775 bytes32 constant ADMIN_ROLE = keccak256("ADMIN_ROLE");
// 0xf4c25f2263d0df187444b70e47283d622c70181c5baeb1306a01edba1ce184c bytes32 constant DEPLOYER_ROLE = keccak256("DEPLOYER_ROLE");
// 0x62150a51582c26f4255242a3c4ca35fb042500e7315069523d55676aed01a56a bytes32 constant TOKEN_OPERATOR_ROLE = keccak256("TOKEN_OPERATOR_ROLE");
// 0xbc34dca9375a29f1b549ee19900a8183308a2d43e0192eb541bc5dd4501e bytes32 constant STAKE_OPERATOR_ROLE = keccak256("STAKE_OPERATOR_ROLE");
// 0xf2c4d2c87036c9189c45e222d5dff8eaffb4966ee49e36b4ffca88a2d855f890 bytes32 constant PRICE_FEED_OPERATOR_ROLE = keccak256("PRICE_FEED_OPERATOR_ROLE");
// 0x7373a9c45f77d802b976943bfbcbf6ee8e48e5484c985358187657fl1d28c3f3c603e bytes32 constant PAIR_OPERATOR_ROLE = keccak256("PAIR_OPERATOR_ROLE");
// 0xfbc8737ab85eb45125971625a9ebeb75cc78e01d5c1fa80c4c6e520f47bc4f9f bytes32 constant KEEPER_ROLE = keccak256("KEEPER_ROLE");
// 0x73d877a9d9791a904e418f90a2f248aa2c5f1348317792a6f6412f949d9f819f7 bytes32 constant PRICE_FEEDER_ROLE = keccak256("PRICE_FEEDER_ROLE");

The preimage string for a role usually matches the variable name. However, the preimage string for STAKE_OPERATOR_ROLE is STAKE_OPERATOR, without the _ROLE suffix.

Recommendation

Consider using a consistent naming system.

Status

The team has resolved this issue in commit **aa0f18a** and **cec8aa6**.
### Appendix

#### Appendix 1 - Files in Scope

This audit covered the following files in commit **c48d047**:

<table>
<thead>
<tr>
<th>File</th>
<th>SHA-1 hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>contracts/ALP.sol</td>
<td>18e85e053e337b5ce2132f757756a1efb5184c9d</td>
</tr>
<tr>
<td>contracts/dependencies/IWBNB.sol</td>
<td>74ac7509c7eb55b8896d9348c0bc5a81ac3d336c</td>
</tr>
<tr>
<td>contracts/diamond/ApolloX.sol</td>
<td>ac1a634e0f70311543e0d9c077b9ac197a1cd071</td>
</tr>
<tr>
<td>contracts/diamond/facets/AccessControlEnumerableFacet.sol</td>
<td>c556ebb0ab3940b7c2d2db78288b262562f179ff</td>
</tr>
<tr>
<td>contracts/diamond/facets/AlpManagerFacet.sol</td>
<td>187421863437d8c7eebaafecf3f67997252342a8</td>
</tr>
<tr>
<td>contracts/diamond/facets/ApxRewardFacet.sol</td>
<td>2dbcc0451833f9ac6e79513850d5e0ff559c8bd</td>
</tr>
<tr>
<td>contracts/diamond/facets/BrokerManagerFacet.sol</td>
<td>057a7bd70005525f12f217f4e05336d19a476d7a</td>
</tr>
<tr>
<td>contracts/diamond/facets/ChainlinkPriceFacet.sol</td>
<td>dcea0e7b19c8333fe1d782264e3d947f05f711dc</td>
</tr>
<tr>
<td>contracts/diamond/facets/DiamondCutFacet.sol</td>
<td>aa3facac928dcd2b5832ed3262637ac815f58df</td>
</tr>
<tr>
<td>contracts/diamond/facets/DiamondLoupeFacet.sol</td>
<td>f147a500fed8fd6c73eddbfd90a15968d0ff20fe</td>
</tr>
<tr>
<td>contracts/diamond/facets/FeeManagerFacet.sol</td>
<td>9bdcb2f7830720b3311e31c0dd864a2f6c0c60</td>
</tr>
<tr>
<td>contracts/diamond/facets/LimitOrderFacet.sol</td>
<td>8927854e3c1b153335223518aba67d078b0ac71</td>
</tr>
<tr>
<td>contracts/diamond/facets/OrderAndTradeHistoryFacet.sol</td>
<td>fd91f24959b6d2c3a994447d87a42968e8f0c98</td>
</tr>
<tr>
<td>contracts/diamond/facets/PairsManagerFacet.sol</td>
<td>e07f11b28bcf445b0aa540f1b9f9ecbf7484c55d5</td>
</tr>
<tr>
<td>contracts/diamond/facets/PausableFacet.sol</td>
<td>d643e1cb943443fd2391946a99c5f414f14b9b4b</td>
</tr>
<tr>
<td>contracts/diamond/facets/PriceFacadeFacet.sol</td>
<td>8a30da181fbec2acfac3b8d7c22efa81d575f1d</td>
</tr>
<tr>
<td>contracts/diamond/facets/StakeRewardFacet.sol</td>
<td>bdbbccb6e27f7e7124fe7d65c03a2c3b6f8c351a</td>
</tr>
<tr>
<td>contracts/diamond/facets/TradingCheckerFacet.sol</td>
<td>de32f9a0a443379a63d7720a83de1aa90b35f4</td>
</tr>
<tr>
<td>contracts/diamond/facets/TradingCloseFacet.sol</td>
<td>74b54ff72ea2b8203734c6f69263239ee0a287</td>
</tr>
<tr>
<td>contracts/diamond/facets/TradingConfigFacet.sol</td>
<td>92861d40b3962e0643f494aca1e67b21c93571f60</td>
</tr>
<tr>
<td>contracts/diamond/facets/TradingCoreFacet.sol</td>
<td>2c5fb40b8845a53d3deaaab8e832e5a90053908</td>
</tr>
<tr>
<td>contracts/diamond/facets/TradingOpenFacet.sol</td>
<td>d4312f49ceb0fe27dd7643a3e30dc224061583e3</td>
</tr>
<tr>
<td>File Name</td>
<td>Hash Value</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>contracts/diamond/facets/TradingPortalFacet.sol</td>
<td>469c9df522acd59c3da14fb7efb44586a747481</td>
</tr>
<tr>
<td>contracts/diamond/facets/TradingReaderFacet.sol</td>
<td>fc319cbd639b398908b230b75d370986ca924cc8</td>
</tr>
<tr>
<td>contracts/diamond/facets/TransitionFacet.sol</td>
<td>945e3c9ea0d2fcefe5e1b35933e92ae813dabe</td>
</tr>
<tr>
<td>contracts/diamond/facets/VaultFacet.sol</td>
<td>59be54d9be822f5485883c91967a57761e29aa41</td>
</tr>
<tr>
<td>contracts/diamond/interfaces/IAlpManager.sol</td>
<td>f3d780bbcc4c8c5b99905bb7df041ac1b2e44ae9</td>
</tr>
<tr>
<td>contracts/diamond/interfaces/IApxReward.sol</td>
<td>c34a82595b74a4dd6132836d23457e7816d09abe</td>
</tr>
<tr>
<td>contracts/diamond/interfaces/IBook.sol</td>
<td>f7c5daee94406df2de1ef20a669edc78ca5fca56</td>
</tr>
<tr>
<td>contracts/diamond/interfaces/IBrokerManager.sol</td>
<td>24fc0501ad481071870f860da0e0254d40e5109</td>
</tr>
<tr>
<td>contracts/diamond/interfaces/IChainlinkPrice.sol</td>
<td>71b78d23e83b652dd363b044c94840794e2742e</td>
</tr>
<tr>
<td>contracts/diamond/interfaces/IDiamondCut.sol</td>
<td>9489b488d03bb38b78aa7f79b88f8eb7079bb29</td>
</tr>
<tr>
<td>contracts/diamond/interfaces/IDiamondLoupe.sol</td>
<td>86ed93947d546cc23f4ff3a8ff2421ec24b5f9aa</td>
</tr>
<tr>
<td>contracts/diamond/interfaces/IFeeManager.sol</td>
<td>94d8c1e102503e2c3a3a81e6e9a59e2c57a42717</td>
</tr>
<tr>
<td>contracts/diamond/interfaces/ILimitOrder.sol</td>
<td>18b5c686724ba84fde7bbe880a7f53b455d24e92</td>
</tr>
<tr>
<td>contracts/diamond/interfaces/IDOraclePrice.sol</td>
<td>7f5fa0f5971273c8dcf7685bf43a7ff4577380</td>
</tr>
<tr>
<td>contracts/diamond/interfaces/IOrderAndTradeHistory.sol</td>
<td>5ff29edca298b7c8870d5a676a3e2defde51bc0b</td>
</tr>
<tr>
<td>contracts/diamond/interfaces/IPairsManager.sol</td>
<td>5481556fc84e58075e307deff7b7d48e628abb60</td>
</tr>
<tr>
<td>contracts/diamond/interfaces/IPausable.sol</td>
<td>c1ea0caab18fec8075bc07b2466f932046f488c9</td>
</tr>
<tr>
<td>contracts/diamond/interfaces/IPriceFacade.sol</td>
<td>9ed0fe858c44f3b347bb619332778d127f343ba6</td>
</tr>
<tr>
<td>contracts/diamond/interfaces/ISetReward.sol</td>
<td>31cba313d25af4993870b0e4001ea1b594f6204</td>
</tr>
<tr>
<td>contracts/diamond/interfaces/ITrading.sol</td>
<td>2dd243a4804d4dc2c9028e9ecc872f502102becd</td>
</tr>
<tr>
<td>contracts/diamond/interfaces/ITradingChecker.sol</td>
<td>6628d59c216c1b65f07097ca871a2c1c38818e1</td>
</tr>
<tr>
<td>contracts/diamond/interfaces/ITradingClose.sol</td>
<td>390bc102ca81e5f3c8eb8d8c561a7f9133b6e51</td>
</tr>
<tr>
<td>contracts/diamond/interfaces/ITradingConfig.sol</td>
<td>363eb92ddadc3bc52c63d2cb656fdaa6c8c124c2</td>
</tr>
<tr>
<td>contracts/diamond/interfaces/ITradingCore.sol</td>
<td>34df27407088ecc1c5b6e38512d1e48fa0817117</td>
</tr>
<tr>
<td>contracts/diamond/interfaces/ITradingOpen.sol</td>
<td>5eee63f0b25bb28514b78fb9a9e738b4d645c44b</td>
</tr>
<tr>
<td>contracts/diamond/interfaces/ITradingPortal.sol</td>
<td>3c2a30a66c5f97f9624cda1a6ac6af25111d04b39</td>
</tr>
<tr>
<td>contracts/diamond/interfaces/ITradingReader.sol</td>
<td>a98b666f55074e4acd00bde1402ac444c85a159</td>
</tr>
<tr>
<td>contracts/diamond/interfaces/IVault.sol</td>
<td>3bfc87a133b7bb8e7a4f8b503c53e7c35bb8e8d</td>
</tr>
<tr>
<td>contracts/diamond/libraries/LibAccessControlEnumerable.so</td>
<td>364879e6decf6541eef276ef6a8ab0b55943737bl</td>
</tr>
</tbody>
</table>