

LIQ Markets

Smart Contract Security Assessment

Version 2.0

Audit dates: Sep 17 — Sep 26, 2024

Audited by: Ox1771
peakbolt
spicymeatball

Contents

1. Introduction

1.1 About Zenith

1.2 Disclaimer

1.3 Risk Classification

2. Executive Summary

2.1 About LIQ Markets

2.2 Scope

2.3 Audit Timeline

2.4 Issues Found

3. Findings Summary

4. Findings

4.1 High Risk

4.2 Medium Risk

4.3 Low Risk

4.4 Informational

1. Introduction

1.1 About Zenith

Zenith is an offering by Code4rena that provides consultative audits from the very best security researchers in the space. We focus on crafting a tailored security team specifically for the needs of your codebase.

Learn more about us at <https://code4rena.com/zenith>.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2. Executive Summary

2.1 About LIQ Markets

Trade and earn cryptocurrencies with lowest fees, depthless liquidity, and up to 100x leverage. Generate yield in a bull, bear, or sideways market.

2.2 Scope

Repository	Liq-Markets/perp-contracts-audit
Commit Hash	875d212bcacce6590b3cb04eca3686bebfa74807
Mitigation Hash	1293357239049916517

2.3 Audit Timeline

DATE	EVENT
Sep 17, 2024	Audit start
Sep 26, 2024	Audit end
Oct 24, 2024	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	1
Medium Risk	5
Low Risk	3
Informational	3
Total Issues	12

3. Findings Summary

ID	DESCRIPTION	STATUS
H-1	Contracts interacting with `PositionRouter` may lose funds	Resolved

M-1	Incorrect fee calculation due to use of `msg.sender` in `_collectFees` function	Resolved
M-2	`RewardReader.getStakingInfo()` will return the wrong `amounts[]`	Resolved
M-3	`priceFeedSetTokenConfig()` will always fail due to incorrect `action` hash in `signalPriceFeedSetTokenConfig()`	Resolved
M-4	Slippage checked should be enabled for compounding logic in `RewardRouterV2`	Resolved
M-5	`initRewardRouter()` will revert as `stakedGlpTracker()` is no longer in `RewardRouterV2`	Resolved
L-1	Confidence intervals not applied to Pyth feed prices	Resolved
L-2	The RewardReader will return an incorrect token address array in the getStakingInfo function	Resolved
L-3	Missing loop exit when the execution and cancellation of the `decreasePosition` request fail.	Acknowledged
I-1	Unused variables in `VaultPriceFeed.sol`	Resolved
I-2	Missing `referralCode` check for `_emitIncreasePositionReferral()`	Resolved
I-3	Un-used `averageStakedAmounts` in `RewardTracker`	Resolved

4. Findings

4.1 High Risk

A total of 1 high risk findings were identified.

[H-1] Contracts interacting with `PositionRouter` may lose funds

Severity: High

Status: Resolved

Context:

- [BasePositionManager.sol#L285](#)

Description:

The `LiqMarket` protocol uses `_transferOutETHWithGasLimitIgnoreFail` function to send ETH to recipients:

```
function _transferOutETHWithGasLimitIgnoreFail(uint256 _amountOut,
address payable _receiver) internal {
    IWETH(weth).withdraw(_amountOut);

    // use `send` instead of `transfer` to not revert whole
    // transaction in case ETH transfer was failed
    // it has limit of 2300 gas
    // this is to avoid front-running
    >> _receiver.send(_amountOut);
}
```

The `send` function uses only 2300 gas to transfer ETH, which may not be sufficient if the recipient is a contract wallet with custom logic in its receive function. Some Gnosis Safe wallets might require more than 2300 gas for a transaction to succeed. Since there is no check to verify whether the send was successful, the withdrawn ETH could become trapped in the router, causing the receiver to lose their funds.

For example, in a scenario where `executeDecreasePositions` is called and `request.withdrawETH` is set to `true`, the withdrawn ETH will remain in the router.

Recommendation:

It is recommended to use `call` with a specified `gasLimit` to transfer ETH, and fallback to transferring WETH if the ETH transfer fails, similar to GMX:

```
function _transferOutETHWithGasLimitFallbackToWeth(uint256
_amountOut, address payable _receiver) internal {
    IWETH _weth = IWETH(weth);
    _weth.withdraw(_amountOut);

    // re-assign ethTransferGasLimit since only local variables
    // can be used in assembly calls
    uint256 _ethTransferGasLimit = ethTransferGasLimit;

    bool success;
    // use an assembly call to avoid loading large data into memory
    // input mem[in...(in+insize)]
    // output area mem[out...(out+outsize)]
    assembly {
        success := call(
            _ethTransferGasLimit, // gas limit
            _receiver, // receiver
            _amountOut, // value
            0, // in
            0, // insize
            0, // out
            0 // outsize
        )
    }

    if (success) { return; }

    // if the transfer failed, re-wrap the token and send it to the
    receiver
    _weth.deposit{ value: _amountOut }();
    _weth.transfer(address(_receiver), _amountOut);
}
```

As a compromise it is possible to keep `send` and transfer WETH in case of failure:

```
function _transferOutETHWithGasLimitIgnoreFail(uint256 _amountOut,
address payable _receiver) internal {
    IWETH(weth).withdraw(_amountOut);

    // use `send` instead of `transfer` to not revert whole
    transaction in case ETH transfer was failed
```

```
        // it has limit of 2300 gas
        // this is to avoid front-running
+       bool success = _receiver.send(_amountOut);
+       if(!success) {
+           _weth.deposit{ value: _amountOut }();
+           _weth.transfer(address(_receiver), _amountOut);
+       }
    }
```

LIQ Markets:

Fixed in the following [commit](#)

Zenith:

Verified

4.2 Medium Risk

A total of 5 medium risk findings were identified.

[M-1] Incorrect fee calculation due to use of `msg.sender` in `_collectFees` function

Severity: Medium

Status: Resolved

Context:

- [PositionRouter.sol#L437](#)

Description:

When a user's `increasePosition` request is executed, he should pay additional fees based on the status of his existing position:

```
function _collectFees(
    address _account,
    address[] memory _path,
    uint256 _amountIn,
    address _indexToken,
    bool _isLong,
    uint256 _sizeDelta
) internal returns (uint256) {
    bool shouldDeductFee = _shouldDeductFee(
>>         _account,
            _path,
            _amountIn,
            _indexToken,
            _isLong,
            _sizeDelta
    );
}
```

However, when the `_collectFees` function is called within `executeIncreasePosition`, `msg.sender` is used instead of `account.request`. This causes the fee calculation to be based on the caller's account data, not the original request creator's. As a result, fees could be applied incorrectly.

Recommendation:

```
function executeIncreasePosition(bytes32 _key, address payable
_executionFeeReceiver) public nonReentrant returns (bool) {
    ---SNIP---
    +         uint256 afterFeeAmount = _collectFees(request.account,
request.path, amountIn, request.indexToken, request.isLong,
request.sizeDelta);
```

LIQ Markets:

Fixed in the following [commit](#)

Zenith:

Verified

[M-2] `RewardReader.getStakingInfo()` will return the wrong `amounts[]`

Severity: Medium

Status: Resolved

Context:

- [RewardReader.sol#L27-L53](#)

Description:

`getStakingInfo()` will return the `amounts[]` for the properties in the reward trackers.

However, the index `amounts[i * propsLength + j * propsLength]` is incorrect as `i*propsLength` will overlap when `i > 0`.

Suppose we have 2 rewardTrackers, and the first rewardTracker has 2 rewardTokens. When `i=0, j=0`, indexes will be 0 to 4 When `i=0, j=1`, indexes will be 5 to 9 When `i=1, j=0`, indexes will be 5 to 9 (This is incorrect as it overlaps with previous indexes)

```
function getStakingInfo(address _account, address[] memory
_rewardTrackers) public view returns (uint256[] memory, address[] memory)
{
    uint256 propsLength = 5;
    uint256 totalPropsLength = 0;

    for (uint256 i = 0; i < _rewardTrackers.length; i++) {
        address rewardDistributor =
        IRewardTrackerExtended(_rewardTrackers[i]).distributor();
        totalPropsLength = totalPropsLength + (
        IRewardDistributor(rewardDistributor).allRewardTokensLength()*
        propsLength);
    }
    address[] memory rewardTokenAddresses = new address[]
    (totalPropsLength/propsLength);
    uint256[] memory amounts = new uint256[](totalPropsLength);
    for (uint256 i = 0; i < _rewardTrackers.length; i++) {
        IRewardTracker rewardTracker =
        IRewardTracker(_rewardTrackers[i]);
        address[] memory rewardTokens =
        IRewardTracker(_rewardTrackers[i]).getAllRewardTokens();

        for (uint256 j = 0; j < rewardTokens.length; j++) {
            address rewardToken = rewardTokens[j];
```

```

        rewardTokenAddresses[rewardTokenAddresses.length] =
rewardToken;
>>>        amounts[i * propsLength + j * propsLength] =
rewardTracker.claimable(_account, rewardToken);
>>>        amounts[i * propsLength + j * propsLength + 1] =
rewardTracker.tokensPerInterval(rewardToken);
>>>        amounts[i * propsLength + j * propsLength + 2] =
rewardTracker.averageStakedAmounts(_account);
>>>        amounts[i * propsLength + j * propsLength + 3] =
rewardTracker.cumulativeRewards(_account, rewardToken);
>>>        amounts[i * propsLength + j * propsLength + 4] =
IERC20(rewardToken).totalSupply();
    }
}
return (amounts, rewardTokenAddresses);
}

```

Recommendation:

It should be something like this:

```

    for (uint256 i = 0; i < _rewardTrackers.length; i++) {
        IRewardTracker rewardTracker =
IRewardTracker(_rewardTrackers[i]);

        address[] memory rewardTokens =
IRewardTracker(_rewardTrackers[i]).getAllRewardTokens();
+         uint256 offset = 0;

        for (uint256 j = 0; j < rewardTokens.length; j++) {

            address rewardToken = rewardTokens[j];
            rewardTokenAddresses[rewardTokenAddresses.length] =
rewardToken;
-             amounts[i * propsLength + j * propsLength] =
rewardTracker.claimable(_account, rewardToken);
-             amounts[i * propsLength + j * propsLength + 1] =
rewardTracker.tokensPerInterval(rewardToken);
-             amounts[i * propsLength + j * propsLength + 2] =
rewardTracker.averageStakedAmounts(_account);
-             amounts[i * propsLength + j * propsLength + 3] =
rewardTracker.cumulativeRewards(_account, rewardToken);

```

```

-         amounts[i * propsLength + j * propsLength + 4] =
IERC20(rewardToken).totalSupply();

+         amounts[offset + j * propsLength] =
rewardTracker.claimable(_account, rewardToken);
+         amounts[offset + j * propsLength + 1] =
rewardTracker.tokensPerInterval(rewardToken);
+         amounts[offset + j * propsLength + 2] =
rewardTracker.averageStakedAmounts(_account);
+         amounts[offset + j * propsLength + 3] =
rewardTracker.cumulativeRewards(_account, rewardToken);
+         amounts[offset + j * propsLength + 4] =
IERC20(rewardToken).totalSupply();

+         if (j == rewardTokens.length - 1) offset += j *
propsLength + 4;
        }
    }
}

```

Sponsor:

Fixed in the following [commit](#)

Zenith:

Verified.

[M-3] `priceFeedSetTokenConfig()` will always fail due to incorrect `action` hash in `signalPriceFeedSetTokenConfig()`

Severity: Medium

Status: Resolved

Context:

- [LiqTimelock.sol#L468-L485](#)
- [LiqTimelock.sol#L63-L69](#)

Description:

The `action` hash for `signalPriceFeedSetTokenConfig()` is created using `pyth` variables instead of chainlink variables. This will cause a mismatch for the `priceFeedSetTokenConfig()`, preventing it from working. The `emit SignalPriceFeedSetTokenConfig()` is also incorrect.

```
function signalPriceFeedSetTokenConfig(
    address _vaultPriceFeed,
    address _token,
    bool _isStrictStable,
    bytes32 _pythPriceId,
    uint256 _pythConfScalingFactor
) external onlyAdmin {
    bytes32 action = keccak256(abi.encodePacked(
        "priceFeedSetTokenConfig",
        _vaultPriceFeed,
        _token,
        _isStrictStable,
        _pythPriceId,
        _pythConfScalingFactor
    ));

    _setPendingAction(action);

    emit SignalPriceFeedSetTokenConfig(
        _vaultPriceFeed,
        _token,
        _isStrictStable,
        _pythPriceId,
        _pythConfScalingFactor
    );
}
```

Recommendation:

```
function signalPriceFeedSetTokenConfig(
    address _vaultPriceFeed,
    address _token,
-    bool _isStrictStable,
-    bytes32 _pythPriceId,
-    uint256 _pythConfScalingFactor
+    address _priceFeed,
+    uint256 _priceDecimals,
+    bool _isStrictStable
) external onlyAdmin {
    bytes32 action = keccak256(abi.encodePacked(
        "priceFeedSetTokenConfig",
        _vaultPriceFeed,
        _token,
-        _isStrictStable,
-        _pythPriceId,
-        _pythConfScalingFactor
+        _priceFeed,
+        _priceDecimals,
+        _isStrictStable
    ));

    _setPendingAction(action);

    emit SignalPriceFeedSetTokenConfig(
        _vaultPriceFeed,
        _token,
-        _isStrictStable,
-        _pythPriceId,
-        _pythConfScalingFactor
+        _priceFeed,
+        _priceDecimals,
+        _isStrictStable
    );
}
```

LIQ Markets: Fixed in the following [commit](#)

Zenith: Verified

[M-4] Slippage checked should be enabled for compounding logic in `RewardRouterV2`

Severity: Medium

Status: Resolved

Context:

- [RewardRouterV2.sol#L142-L147](#)
- [RewardRouterV2.sol#L173-L178](#)

Description:

In both `handleRewards()` and `claim()`, they will call `_mintAndStakeLlpEth()` and `_mintAndStakeLlp()` when compound is enabled.

However, `_minUsdl` and `_minLlp` are both set to zero, which means the slippage protection for `ILlpManager(llpManager).addLiquidityForAccount()` will be disabled. This issue will make the compound logic vulnerable to sandwich attacks, causing the user to incur losses on the compounded rewards.

```
function claim(address _rewardToken, bool _compound, bool
_withdrawETH) external nonReentrant {
    require(IRewardTracker(feeLlpTracker).allTokens(_rewardToken),
"RewardRouter: not _rewardToken"); // TODO check against token if reward
token exist
    address account = msg.sender;
    if(_compound && IVault(vault).whitelistedTokens(_rewardToken)) {
        uint256 amount =
IRewardTracker(feeLlpTracker).claimForAccount(account, _rewardToken,
address(this));
        if (amount > 0) {
            if(_rewardToken == weth) {
>>>                _mintAndStakeLlpEth(amount, 0, 0);
            } else {
                IERC20(_rewardToken).approve(llpManager, amount);
>>>                _mintAndStakeLlp(address(this), account,
_rewardToken, amount, 0, 0);
            }
        }
    }
    ...

function handleRewards(
```



```

        bool _shouldConvertWethToEth,
        bool _shouldCompound
    ) external nonReentrant {
        address account = msg.sender;

        if (_shouldConvertWethToEth || _shouldCompound ) {
            (address[] memory tokens,uint256[] memory amounts) =
            IRewardTracker(feeLlpTracker).claimAllForAccount(account, address(this));
            for (uint256 i = 0; i < tokens.length; i++) {
                address token = tokens[i];
                uint256 amount = amounts[i];
                if(amount > 0){
                    if(_shouldCompound &&
IVault(vault).whitelistedTokens(token)){
                        if(token == weth){
>>>                            _mintAndStakeLlpEth(amount,0,0);
                        }else{
>>>                            IERC20(token).approve(llpManager, amount);
                            _mintAndStakeLlp(address(this),account,token,amount,0,0);
                        }
                    }else if(_shouldConvertWethToEth && token == weth ){
                        IWETH(weth).withdraw(amount);
                        payable(account).sendValue(amount);
                    }else{
                        IERC20(token).safeTransfer(account, amount);
                    }
                }
            }
        }
    }
    ...

```

Recommendation:

Ensure that the parameters `_minUsd1` and `_minLlp` are set accordingly by passing them in and not set to zero.

LIQ Markets:

Fixed in [commit](#)

Zenith:

Verified

[M-5] `initRewardRouter()` will revert as `stakedGlpTracker()` is no longer in `RewardRouterV2`

Severity: Medium

Status: Resolved

Context:

- [Timelock.sol#L143](#)

Description:

`stakedGlpTracker()` is no longer in `RewardRouterV2`. This will make the `setHandler()` call revert, causing `initRewardRouter()` to fail.

```
function initRewardRouter() external onlyAdmin {
    IRewardRouterV2 _rewardRouter = IRewardRouterV2(rewardRouter);

    IHandlerTarget(_rewardRouter.feeGlpTracker()).setHandler(rewardRouter,
true);
    //@audit this call will fail as it has been removed

    IHandlerTarget(_rewardRouter.stakedGlpTracker()).setHandler(rewardRouter,
true);
    IHandlerTarget(glpManager).setHandler(rewardRouter, true);
}
```

Recommendation:

It should be removed from `initRewardRouter()` as follows.

```
function initRewardRouter() external onlyAdmin {
    IRewardRouterV2 _rewardRouter = IRewardRouterV2(rewardRouter);

    IHandlerTarget(_rewardRouter.feeGlpTracker()).setHandler(rewardRouter,
true);
    -
    IHandlerTarget(_rewardRouter.stakedGlpTracker()).setHandler(rewardRouter,
true);
    IHandlerTarget(glpManager).setHandler(rewardRouter, true);
}
```

LIQ Markets:

Fixed in the following [commit](#)

C4 Zenith:

Verified

4.3 Low Risk

A total of 3 low risk findings were identified.

[L-1] Confidence intervals not applied to Pyth feed prices

Severity: Low

Status: Resolved

Context:

- [PythToChainlinkWrapper.sol#L33](#)
- [PythToChainlinkWrapper.sol#L45](#)

Description:

When fetching Pyth prices, `ignoreConfidence` is always set to `true`, which prevents the feed from using the confidence intervals returned in `priceData.conf`:

```
function _getPythPrice(bool _ignoreConfidence, bool _maximise)
internal view returns (uint256, uint80) {
    PythStructs.Price memory priceData = _getPythPriceData();
    uint256 price;
    uint80 roundId;
    // TODO: Check what factor of the confidence interval we want to
use
>>    if(_ignoreConfidence) {
        price = uint256(uint64(priceData.price));
    } else {
        uint256 scaledConf =
uint256(uint64(priceData.conf)).mul(pythConfScalingFactor).div(
PYTH_CONF_SCALING_FACTOR_PRECISION);
        price = _maximise ?
uint256(uint64(priceData.price)).add(scaledConf) :
uint256(uint64(priceData.price)).sub(scaledConf);
    }
```

The Pyth price feed includes a confidence interval (`priceData.conf`) that accounts for potential discrepancies between the returned price and the actual asset price. However, by ignoring this interval, the protocol may miss an opportunity to adjust the price in its favor and allows users to gain advantage via arbitrage.

Recommendation:

Although `VaultPriceFeed` already adjusts asset price using `adjustmentBps` and `_spreadBasisPoints` it is still recommended to include confidence intervals in price calculation.

Sponsor:

Fixed in the following [commit](#)

Zenith:

Verified

[L-2] The RewardReader will return an incorrect token address array in the getStakingInfo function

Severity: Low

Status: Resolved

Context:

[RewardReader.sol#L44](#)

Description:

The `getStakingInfo` function returns `amounts` array with packed data from reward trackers and `rewardTokenAddresses` array containing a list of reward tokens:

```
function getStakingInfo(address _account, address[] memory
_rewardTrackers) public view returns (uint256[] memory, address[] memory)
{
    uint256 propsLength = 5;
    uint256 totalPropsLength = 0;

    for (uint256 i = 0; i < _rewardTrackers.length; i++) {
        address rewardDistributor =
        IRewardTrackerExtended(_rewardTrackers[i]).distributor();
        totalPropsLength = totalPropsLength + (
        IRewardDistributor(rewardDistributor).allRewardTokensLength()*
        propsLength);
    }
    address[] memory rewardTokenAddresses = new address[]
    (totalPropsLength/propsLength);
    uint256[] memory amounts = new uint256[](totalPropsLength);
    for (uint256 i = 0; i < _rewardTrackers.length; i++) {
        IRewardTracker rewardTracker =
        IRewardTracker(_rewardTrackers[i]);
        address[] memory rewardTokens =
        IRewardTracker(_rewardTrackers[i]).getAllRewardTokens();

        for (uint256 j = 0; j < rewardTokens.length; j++) {

            address rewardToken = rewardTokens[j];
            rewardTokenAddresses[rewardTokenAddresses.length] =
            rewardToken;
        }
    }
}
```

However, the issue is that only the element at the index `rewardTokenAddresses.length` will be populated, while all other elements remain empty.

Recommendation:

```
+      uint256 tokenCounter = 0;
      for (uint256 i = 0; i < _rewardTrackers.length; i++) {
          IRewardTracker rewardTracker =
IRewardTracker(_rewardTrackers[i]);
          address[] memory rewardTokens =
IRewardTracker(_rewardTrackers[i]).getAllRewardTokens();

          for (uint256 j = 0; j < rewardTokens.length; j++) {

              address rewardToken = rewardTokens[j];
+              rewardTokenAddresses[tokenCounter++] = rewardToken;
```

LIQ Markets:

Fixed in the following [commit](#)

Zenith:

Verified, `offset` is used to track the index of tokens in the `rewardTokenAddresses` array.

[L-3] Missing loop exit when the execution and cancellation of the `decreasePosition` request fail.

Severity: Low

Status: Acknowledged

Context:

[PositionRouter.sol#L288](#)

Description:

When the `keeper` executes a batch of `decreasePosition` requests, the `PositionRouter` starts with `decreasePositionRequestKeysStart`:

```
function executeDecreasePositions(uint256 _endIndex, address payable
_executionFeeReceiver) external override onlyPositionKeeper {
    uint256 index = decreasePositionRequestKeysStart;
    ---SNIP---
    while (index < _endIndex) {
        bytes32 key = decreasePositionRequestKeys[index];

        try this.executeDecreasePosition(key, _executionFeeReceiver)
returns (bool _wasExecuted) {
            if (!_wasExecuted) { break; }
        } catch {
            // wrap this call in a try catch to prevent invalid
cancels from blocking the loop
            try this.cancelDecreasePosition(key,
_executionFeeReceiver) returns (bool _wasCancelled) {
                if (!_wasCancelled) { break; }
            } catch {}
        }

        delete decreasePositionRequestKeys[index];
        index++;
    }

    decreasePositionRequestKeysStart = index;
}
```

In case both execution and cancellation of a request fail, the loop continues to the next request, deleting the failed one. This can potentially trap the user's funds.

Recommendation:

it's recommended to add a break within the cancel function's `try-catch` block to halt the loop if both execution and cancellation fail.

LIQ Markets:

If we put a break in the catch block, it will prevent the batch from being executed, and one bad debt could create a series of pending decrease requests - Acknowledged.

Zenith:

Although the probability of a `cancelDecreasePosition` revert is low, this issue can be ignored for safety reasons.

4.4 Informational

A total of 3 informational findings were identified.

[I-1] Unused variables in `VaultPriceFeed.sol`

Severity: Informational

Status: Resolved

Context:

[VaultPriceFeed.sol#L96](#) [VaultPriceFeed.sol#L100](#) [VaultPriceFeed.sol#L104](#)

Description:

The `VaultPriceFeed.sol` contract includes setters for `spreadThresholdBasisPoints`, `favorPrimaryPrice` and `priceSampleSpace`. However, these variables are not used anywhere in the code.

Recommendation:

If these variables serve no purpose, it is recommended to remove them and their setters from the contract.

LIQ Markets:

Fixed with the following [commit](#)

Zenith:

Verified

[I-2] Missing `referralCode` check for `_emitIncreasePositionReferral()`

Severity: Informational

Status: Resolved

Context:

- [BasePositionManager.sol#L223-L238](#)

Description:

`_emitIncreasePositionReferral()` is missing the `referralCode == bytes32(0)` check, which will cause it to emit the event even when `referralCode` is not set. This is inconsistent with `_emitDecreasePositionReferral()`, which does perform the check.

```
function _emitIncreasePositionReferral(address _account, uint256
_sizeDelta) internal {
    address _referralStorage = referralStorage;
    if (_referralStorage == address(0)) {
        return;
    }

    (bytes32 referralCode, address referrer) =
    IReferralStorage(_referralStorage).getTraderReferralInfo(_account);
    emit IncreasePositionReferral(
        _account,
        _sizeDelta,
        IVault(vault).marginFeeBasisPoints(),
        referralCode,
        referrer
    );
}
```

Recommendation:

This can be resolved by as follows,

```
function _emitIncreasePositionReferral(address _account, uint256
_sizeDelta) internal {
    address _referralStorage = referralStorage;
    if (_referralStorage == address(0)) {
        return;
    }
}
```

```

        (bytes32 referralCode, address referrer) =
        IReferralStorage(_referralStorage).getTraderReferralInfo(_account);

+         if (referralCode == bytes32(0)) {
+             return;
+         }

        emit IncreasePositionReferral(
            _account,
            _sizeDelta,
            IVault(vault).marginFeeBasisPoints(),
            referralCode,
            referrer
        );
    }

```

LIQ Markets:

Fixed in the following [commit](#)

Zenith:

Verified

[I-3] Un-used `averageStakedAmounts` in `RewardTracker`

Severity: Informational

Status: Resolved

Context:

- [RewardTracker.sol#L429-L431](#)

Description:

`averageStakedAmounts` is currently un-used.

Within `_updateAccountRewards()`, it is also incorrectly updated as it mixes the global average with cumulative rewards for diff reward tokens.

```
        averageStakedAmounts[_account] =
        averageStakedAmounts[_account].mul(cumulativeRewards[_account]
[_rewardToken]).div(nextCumulativeReward).add(
            stakedAmount.mul(accountReward).div(nextCumulativeReward)
        );
```

Recommendation:

This can be resolved by removing the code for `averageStakedAmounts`.

LIQ Markets:

Fixed in the following [commit](#)

Zenith:

Verified