Report

v. 2.0

Customer

Reya Labs

Smart Contract Audit

Reya Network. Part II

5th April 2024

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	04.04.24	A. Zveryanskaya	Initial Draft
0.2	04.04.24	A. Zveryanskaya	Minor revision
1.0	04.04.24	A. Zveryanskaya	Release
1.1	05.04.24	A. Zveryanskaya	CVF-5, 8, 15 downgraded
2.0	05.04.24	A. Zveryanskaya	Release

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting (Mikhail Vladimirov and Dmitry Khovratovich) at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.

Reya Network is a trading-optimised modular L2 that focuses on three pillars of performance, liquidity and capital efficiency.

3 Project scope

We were asked to review the code related to the perp and pool of the Reya Network. Corresponding fixes were provided in the Part III of the Audit.

Files:

passive-pool/		
PassivePoolProxy.sol		
passive-pool/storage/		
ShareBalances.sol	PoolIdStore.sol	Pool.sol
GlobalConfiguration.sol		
passive-pool/modules/		
ConfigurationModule.sol	ERC721Receiver Module.sol	FeatureFlagModule.sol
OwnerUpgrade Module.sol	SharesModule.sol	
passive-pool/libraries/		
Errors.sol	Events.sol	FeatureFlagSupport.sol
passive-pool/interfaces/		
IConfigurationModule.sol	ISharesModule.sol	
passive-perp/		
PassivePerpProxy.sol		
passive-perp/storage/		
GlobalConfiguration.sol	Market.sol	MarketConfiguration.sol
PerpPositions.sol	RebateConfiguration.sol	TakerFee Configuration.sol
passive-perp/modules/		
ConfigurationModule.sol	FeatureFlagModule.sol	OwnerUpgrade Module.sol
PassivePerpInstrument Module.sol		

passive-perp/libraries/		
QuadraticEquation.sol	Prices.sol	PerpPositionSupport.sol
Permissions.sol	MaxExposure.sol	MatchOrders.sol
Liquidations.sol	FundingRate.sol	FeatureFlagSupport.sol
Events.sol	Errors.sol	DataTypes.sol
Constants.sol		
passive-perp/interfaces/		
IPassivePerpInformation Module.sol	IConfigurationModule.sol	

4 Methodology

The methodology is not a strict formal procedure, but rather a selection of methods and tactics combined differently and tuned for each particular project, depending on the project structure and technologies used, as well as on client expectations from the audit.

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows best code practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places as well as their visibility scopes and access levels are relevant. At this phase, we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and done properly. At this phase, we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check if code actually does what it is supposed to do, if that algorithms are optimal and correct, and if proper data types are used. We also make sure that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

We classify issues by the following severity levels:

- **Critical issue** directly affects the smart contract functionality and may cause a significant loss.
- **Major issue** is either a solid performance problem or a sign of misuse: a slight code modification or environment change may lead to loss of funds or data. Sometimes it is an abuse of unclear code behaviour which should be double checked.
- **Moderate issue** is not an immediate problem, but rather suboptimal performance in edge cases, an obviously bad code practice, or a situation where the code is correct only in certain business flows.
- **Minor issues** contain code style, best practices and other recommendations.

5 Our findings

We found 13 major, and a few less important issues. All identified Major issues have been fixed.

	Info	Fixed
Major	0	13
Moderate	5	1

Fixed 14 out of 19 issues

6 Major Issues

CVF-1 FIXED

- **Category** Unclear behavior
- **Source** GlobalConfiguration.sol

Description Unlike other similar storage addresses, this address is a plain string rather than the hash of a string.

Recommendation Consider using the hash of a string for consistency.

Client Comment *Replaced with "keccak256(bytes("xyz.reya.....″));"*

```
24 bytes32 s = "xyz.reya.GlobalConfiguration";
```

CVF-2 FIXED

- **Category** Suboptimal
- **Source** ShareBalances.sol

Description Using the "abi.encode" function with a string argument is suboptimal.

Recommendation Consider using "abi.encodePacked" instead. Also consider replacing the string with its hash.

```
30 bytes32 s = keccak256(abi.encode("xyz.reya.ShareBalances", poolId));
```

CVF-3 FIXED

- **Category** Suboptimal
- **Source** Pool.sol

Description Using the "abi.encode" function with a string argument is suboptimal.

Recommendation Consider using "abi.encodePacked" instead. Also consider replacing the string with its hash.

```
53 bytes32 s = keccak256(abi.encode("xyz.reya.Pool", id));
```

CVF-4 FIXED

- **Category** Suboptimal
- **Source** PerpPositions.sol

Description Using the “abi.encode” function with a string argument is suboptimal.

Recommendation Consider using “abi.encodePacked” instead. Also consider replacing the string with its hash.

```
29 bytes32 s = keccak256(abi.encode("xyz.reya.PerpPositions", marketId)
     ↪ );
```

CVF-6 FIXED

- **Category** Suboptimal
- **Source** PerpPositions.sol

Recommendation This code is redundant. Just use the position reference returned by the ”update” call above.

```
93 Market.Data storage market = Market.exists(marketId);
   PerpPosition memory position = load(marketId).positions[accountId];
```

CVF-7 FIXED

- **Category** Suboptimal
- **Source** MarketConfiguration.sol

Description Using the “abi.encode” function with a string argument is suboptimal.

Recommendation Consider using “abi.encodePacked” instead. Also consider replacing the string with its hash.

```
19 bytes32 s = keccak256(abi.encode("xyz.reya.MarketConfiguration",
     ↪ marketId));
```

CVF-9 FIXED

- **Category** Suboptimal
- **Source** Market.sol

Description Using the “abi.encode” function with a string argument is suboptimal.

Recommendation Consider using “abi.encodePacked” instead. Also consider replacing the string with its hash.

```
79 bytes32 s = keccak256(abi.encode("xyz.reya.Market", marketId));
```

CVF-10 FIXED

- **Category** Unclear behavior
- **Source** Market.sol

Description This should be executed only when ”msg.sender” is not the owner.

```
98 address coreProxyAddress = GlobalConfiguration.getCoreProxyAddress()
   ↪ ;
```

CVF-11 FIXED

- **Category** Suboptimal
- **Source** TakerFeeConfiguration.sol

Description Using the “abi.encode” function with a string argument is suboptimal.

Recommendation Consider using “abi.encodePacked” instead. Also consider replacing the string with its hash.

```
21 bytes32 s = keccak256(abi.encode("xyz.reya.TakerFeeConfiguration",
   ↪ marketId));
```

CVF-12 FIXED

- **Category** Suboptimal
- **Source** Prices.sol

Description This check optimizes a rare case of zero order base at a cost of making the most common case more expensive.

Recommendation Consider removing this check.

```
41 if (orderBase.eq(ZERO_sd)) {
       return;
```

CVF-13 FIXED

- **Category** Documentation
- **Source** PerpPositionSupport.sol

Description The comment doesn't match the code. In the code tracker delta is actually divided by the last base multiplier, not the last ADL exposure multiplier. Maybe just confusing terminoligy, in which case consider using consistent terminology across code and comments.

Client Comment *Changed comment to "ADL base multiplier".*

```
30 // divide the tracker delta by the last ADL exposure multiplier
   SD59x18 lastBaseMultiplier = position.trackers.baseMultiplier.
      ↪ intoSD59x18();
   SD59x18 fundingFeeDelta = trackerDelta.div(lastBaseMultiplier);
```

CVF-14 FIXED

- **Category** Overflow/Underflow
- **Source** MaxExposure.sol

Recommendation It would be more efficient and less prone to phantom overflow, to calculate C as: lmr^2-(balance/lmrMultiplier)^2

```
29 cWad = lmrSquared.sub(balanceSquared.div(imrMultiplierSquared));
```

CVF-16 FIXED

- **Category** Procedural
- **Source** Events.sol

Recommendation ID parameters, such as ”marketId”, ”accountId”, ”exchangeId” etc should be indexed.

```
39 event MarketConfigurationUpdated(uint128 marketId,
   ↪ MarketConfigurationData marketConfig, uint256 blockTimestamp);

48 event PositionDataUpdated(uint128 marketId, uint128 accountId,
   ↪ PerpPosition positionData, uint256 blockTimestamp);

60     uint128 marketId,
       uint128 accountId,

79     uint128 marketId,
80     uint128 liquidatableAccountId,

82     uint128 liquidatorAccountId,

94 event ExchangeRebateUpdated(uint128 exchangeId, UD60x18 rebate,
   ↪ uint256 blockTimestamp);

102 event PoolRebateUpdated(uint128 poolId, UD60x18 rebate, uint256
   ↪ blockTimestamp);

111 event TierFeeUpdated(uint128 marketId, uint256 tierId, UD60x18
   ↪ feeParameter, uint256 blockTimestamp);

120 event AccountTierUpdated(uint128 marketId, uint128 accountId,
   ↪ uint256 tierId, uint256 blockTimestamp);
```

7 Moderate Issues

CVF-5 INFO

- **Category** Suboptimal
- **Source** PerpPositions.sol

Description Here position data, just written into the storage is read back from the storage.

Recommendation Consider using the written position data.

Client Comment *The function returns storage position and getLatest returns memory position. We have to save the position in memory then return it.*

```
71 load(marketId).positions[accountId] = getLatest(marketId, accountId)
      ↪ ;
```

```
77     positionData: position,
```

CVF-8 INFO

- **Category** Unclear behavior
- **Source** Market.sol

Description This allows loading a market with zero ID, while zero ID is reserved to mark non-exiting markets.

Recommendation Consider explicitly forbidding loading zero ID market.

Client Comment *This is a private function. 'exists' function forbids this.*

```
78 function load(uint128 marketId) private pure returns (Data storage
      ↪ market) {
```

CVF-15 INFO

- **Category** Suboptimal
- **Source** Events.sol

Description Block timestamp parameters are redundant as every emitted event is bound to a block that has a timestamp.

Recommendation Consider removing timestamp parameters from events.

Client Comment *For the off-chain infrastructure it is easier to get the timestamp as an event parameter. It helps avoid a second rpc call to query the timestamp of the block.*

```
24 event GlobalConfigurationUpdated(GlobalConfiguration.Data
   ↪ globalConfig, uint256 blockTimestamp);

31 event MarketDataUpdated(Market.Data marketData, uint256
   ↪ blockTimestamp);

39 event MarketConfigurationUpdated(uint128 marketId,
   ↪ MarketConfigurationData marketConfig, uint256 blockTimestamp);

48 event PositionDataUpdated(uint128 marketId, uint128 accountId,
   ↪ PerpPosition positionData, uint256 blockTimestamp);

65     uint256 blockTimestamp

85     uint256 blockTimestamp

94 event ExchangeRebateUpdated(uint128 exchangeId, UD60x18 rebate,
   ↪ uint256 blockTimestamp);

102 event PoolRebateUpdated(uint128 poolId, UD60x18 rebate, uint256
   ↪ blockTimestamp);

111 event TierFeeUpdated(uint128 marketId, uint256 tierId, UD60x18
   ↪ feeParameter, uint256 blockTimestamp);

120 event AccountTierUpdated(uint128 marketId, uint128 accountId,
   ↪ uint256 tierId, uint256 blockTimestamp);
```

CVF-17 INFO

- **Category** Procedural
- **Source** Market.sol

Description The comment and the code don't match. Probably just different terminology.

Recommendation Consider making them consistent.

```
394 // compute the price delta wrt. the market price
    SD59x18 adlDeltaPrice = supportedUPnL.div(netBase);
```

CVF-18 FIXED

- **Category** Unclear behavior
- **Source** PassivePerpInstrumentModule.sol

Description In case of unknown liquidation type this function silently does nothing.

Recommendation Consider reverting in such a case.

```
181 LiquidationType liquidationType,
```

CVF-19 INFO

- **Category** Overflow/Underflow
- **Source** PerpPositionSupport.sol

Description Underflow is possible here.

Recommendation Consider either properly handling it or clearly explaining why it is not possible.

Client Comment *BaseMultiplier should not end up > 1. It represents the non-ADL-ed percentage of a position. (starting from UNIT reducing in size as ADL happens).*

```
54 UD60x18 lastPriceCoefficient = ONE_ud.sub(baseMultiplier);
```

8 Recommendations

CVF-20 INFO

- **Category** Procedural
- **Source** GlobalConfiguration.sol

Recommendation This version requirement could be simplified as "^0.8.19". Also, consider specifying as "^0.8.0" unless there is something special regarding this particular version. This is relevant for the following files: ShareBalances.sol, PoolIdStore.sol, Pool.sol, ConfigurationModule.sol, ERC721ReceiverModule.sol, FeatureFlagModule.sol,6 OwnerUpgradeModule.sol, SharesModule.sol, Events.sol, Errors.sol, FeatureFlagSupport.sol, IConfigurationModule.sol, ISharesModule.sol, PassivePoolProxy.sol, PerpPositions.sol, MarketConfiguration.sol, Market.sol, GlobalConfiguration.sol, RebateConfiguration.sol, TakerFeeConfiguration.sol, ConfigurationModule.sol, FeatureFlagModule.sol, OwnerUpgradeModule.sol, DataTypes.sol, Errors.sol, Prices.sol, PerpPositionSupport.sol, MatchOrders.sol, Liquidations.sol, QuadraticEquation.sol, MaxExposure.sol, FundingRate.sol, Constants.sol, Events.sol, FeatureFlagSupport.sol, Permissions.sol, IConfigurationModule.sol, IPassivePerpInformationModule.sol, PassivePerpProxy.sol.

Client Comment *Same comment as for the core audit.*

```
8 pragma solidity >=0.8.19 <0.9.0;
```

CVF-21 INFO

- **Category** Bad datatype
- **Source** GlobalConfiguration.sol

Recommendation The type for this field should be more specific.

Client Comment *This can only bet set by the owner, so we don't worry about it being misconfigured.*

```
20 address coreProxy;
```

CVF-22 INFO

- **Category** Bad datatype
- **Source** GlobalConfiguration.sol

Recommendation The return type should be more specific.

Client Comment *Aame argument as above.*

```
45 function getCoreProxyAddress() internal view returns (address) {
```

CVF-24 INFO

- **Category** Bad naming
- **Source** PoolIdStore.sol

Recommendation Events are usually named via nouns, such as ”PoolId”.

```
19 event IdAdvanced(uint256 id, uint256 blockTimestamp);
```

CVF-25 FIXED

- **Category** Suboptimal
- **Source** PoolIdStore.sol

Description Here a value just written into the storage is read from the storage back.

Recommendation Consider reusing the written value: id = ++idStore.lastId;

```
40 idStore.lastId += 1;
   id = idStore.lastId;
```

CVF-26 INFO

- **Category** Procedural
- **Source** Pool.sol

Description We didn’t review these files.

```
19 import { IERC20, ERC20Helper } from "@voltz-protocol/util-contracts/
      ↪ src/token/ERC20Helper.sol";
20 import { UD60x18, ud, mulDiv } from "@prb/math/UD60x18.sol";
   import { DecimalMath } from "@voltz-protocol/util-contracts/src/
      ↪ helpers/DecimalMath.sol";
```

CVF-27 INFO

- **Category** Bad datatype
- **Source** Pool.sol

Recommendation The type for this field should be ”IERC20”.

Client Comment *Handling quote tokens addresses is better for our protocol because we track them using SetUtil and rarely casted to ERC20 for transfers.*

```
41 address quoteToken;
```

CVF-28 INFO

- **Category** Suboptimal
- **Source** Pool.sol

Description In ERC20 the ”decimals” property is used by UI to render token amounts in human-readable way. Using this property in smart contracts is discouraged.

Recommendation Consider treating all token amounts are integers.

```
45 uint8 quoteTokenDecimals;
```

CVF-29 INFO

- **Category** Bad datatype
- **Source** Pool.sol

Recommendation The argument type should be ”IERC20”.

Client Comment *See CVF-27.*

```
74 function create(address quoteToken) internal returns (Data storage
   ↪ pool) {
```

CVF-30 INFO

- **Category** Bad datatype
- **Source** Pool.sol

Recommendation The type of the ”quoteToken” argument should be ”IERC20”.

Client Comment *See CVF-27.*

```
121 function coreDeposit(uint128 accountId, address quoteToken, uint256
    ↪ tokenAmount) private {
```

```
146 function coreWithdrawal(uint128 accountId, address quoteToken,
    ↪ uint256 tokenAmount) private {
```

CVF-31 INFO

- **Category** Suboptimal
- **Source** Pool.sol

Description Doing double approve every time looks like waste of gas.

Recommendation Consider doing a simple approve inside a ”try/catch” block and fallback to double approve only in case simple prove failed.

Client Comment *We had issues in the past with ERC20 tokens that were reverting approvals if the approved value was not 0.*

```
125 quoteToken.safeApprove(coreProxy, 0);
```

```
128 quoteToken.safeApprove(coreProxy, tokenAmount);
```

CVF-32 INFO

- **Category** Suboptimal
- **Source** Pool.sol

Description This check is redundant, as it is anyway be performed again inside the ”transferFrom” function. Also, this check doesn’t guarantee successful transfer.

Recommendation Consider removing this check.

Client Comment *The off-chain infrastructure benefits from parsing a standard error for allowance (different tokens have different errors).*

```
186 if (allowance < amount) {
```

CVF-33 INFO

- **Category** Procedural
- **Source** ConfigurationModule.sol

Description We didn't review these files.

```
14 import { OwnableStorage } from "@voltz-protocol/util-contracts/src/
      ↪ storage/OwnableStorage.sol";
   import { SetUtil } from "@voltz-protocol/util-contracts/src/helpers/
      ↪ SetUtil.sol";
```

CVF-34 FIXED

- **Category** Procedural
- **Source** ConfigurationModule.sol

Recommendation Consider wrapping the data, returned by a failed call, into the error.

```
54 revert Errors.CoreCallReverted();
```

CVF-35 INFO

- **Category** Procedural
- **Source** ERC721ReceiverModule.sol

Description We didn't review this file.

```
10 import { IERC721Receiver } from "@voltz-protocol/util-contracts/src/
      ↪ interfaces/IERC721Receiver.sol";
```

CVF-37 INFO

- **Category** Procedural
- **Source** FeatureFlagModule.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
19 contract FeatureFlagModule is BaseFeatureFlagModule { }
```

CVF-39 INFO

- **Category** Procedural
- **Source** OwnerUpgradeModule.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
14 contract OwnerUpgradeModule is BaseOwnerUpgradeModule { }
```

CVF-40 INFO

- **Category** Procedural
- **Source** SharesModule.sol

Description We didn't review this file.

```
16 import { UD60x18 } from "@prb/math/UD60x18.sol";
```

CVF-41 INFO

- **Category** Bad naming
- **Source** Events.sol

Recommendation Events are usually named via nouns, such as "GlobalConfiguration" or "Pool".

```
18 event GlobalConfigurationUpdated(GlobalConfiguration.Data
   ↪ globalConfig, uint256 blockTimestamp);
```

```
26 event PoolCreated(uint128 poolId, address quoteToken, uint128
   ↪ accountId, uint256 blockTimestamp);
```

```
36 event ShareBalanceUpdated(
```

CVF-42 INFO

- **Category** Bad datatype
- **Source** Events.sol

Recommendation The type for the “quoteToken” argument should be “IERC20”.

Client Comment *See CVF-27.*

```
26 event PoolCreated(uint128 poolId, address quoteToken, uint128
   ↪ accountId, uint256 blockTimestamp);
```

CVF-43 FIXED

- **Category** Procedural
- **Source** Errors.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

```
45 error InsufficientTokenOutput();
```

```
60 error CoreCallReverted();
```

CVF-44 INFO

- **Category** Bad datatype
- **Source** IConfigurationModule.sol

Recommendation The argument type should be “IERC20”.

Client Comment *See CVF-27.*

```
37 function createPool(address quoteToken) external returns (uint128
   ↪ poolId);
```

CVF-45 FIXED

- **Category** Documentation
- **Source** IConfigurationModule.sol

Description The semantics of the returned value is unclear.

Recommendation Consider giving a descriptive name to the returned value and/or documenting.

```
47 function sendCallToCore(bytes memory data) external returns (bytes
       ↪ memory);
```

CVF-46 INFO

- **Category** Bad datatype
- **Source** IConfigurationModule.sol

Recommendation The return type should be “IERC20”.

Client Comment *See CVF-27.*

```
54 function getPoolQuoteToken(uint128 id) external view returns (
       ↪ address quoteToken);
```

CVF-47 INFO

- **Category** Procedural
- **Source** ISharesModule.sol

Description We didn’t review this file.

```
10 import { UD60x18 } from "@prb/math/UD60x18.sol";
```

CVF-48 FIXED

- **Category** Unclear behavior
- **Source** ISharesModule.sol

Description It is unclear, how exactly the quote tokens are transferred from the caller to the contract.

Recommendation Consider explaining.

```
18 * @param amount The amount of quote tokens to deposit into the pool.
```

CVF-49 FIXED

- **Category** Documentation
- **Source** ISharesModule.sol

Description The semantics of the returned value is unclear.

Recommendation Consider giving a descriptive name to the returned value and/or documenting.

```
28     returns (uint256);
```

```
37 function removeLiquidity(uint128 poolId, uint256 sharesAmount,
   ↪ uint256 minOut) external returns (uint256);
```

CVF-50 INFO

- **Category** Procedural
- **Source** PassivePoolProxy.sol

Description We didn't review this file.

```
10 import { UUPSProxyWithOwner } from "@voltz-protocol/util-contracts/
   ↪ src/proxy/UUPSProxyWithOwner.sol";
```

CVF-51 INFO

- **Category** Bad datatype
- **Source** PassivePoolProxy.sol

Recommendation The type for this argument should be more specific.

```
18 address firstImplementation,
```

CVF-52 INFO

- **Category** Procedural
- **Source** PassivePoolProxy.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
22 { }
```

CVF-53 INFO

- **Category** Procedural
- **Source** PerpPositions.sol

Description We didn't review these files.

```
17 import { UD60x18 } from "@prb/math/UD60x18.sol";
   import { SD59x18, ZERO as ZERO_sd } from "@prb/math/SD59x18.sol";
```

CVF-54 INFO

- **Category** Suboptimal
- **Source** PerpPositions.sol

Recommendation This line wouldn't be necessary if "position" would be a storage reference.

Client Comment *The 'update' and 'propagateOrderToPerpPosition' return a memory reference.*

```
170 load(marketId).positions[accountId] = position;
```

CVF-55 INFO

- **Category** Procedural
- **Source** Market.sol

Description We didn't review these files.

Client Comment *Some of them were reviewed in audit 1 (unless another team worked in this audit).*

```
35 import { OwnableStorage } from "@voltz-protocol/util-contracts/src/
   ↪ storage/OwnableStorage.sol";
```

```
41 import { DecimalMath } from "@voltz-protocol/util-contracts/src/
   ↪ helpers/DecimalMath.sol";
   import { AccessError } from "@voltz-protocol/util-contracts/src/
   ↪ errors/AccessError.sol";
```

```
54 import { UD60x18, UNIT as ONE_ud, ZERO as ZERO_ud, ud } from "@prb/
   ↪ math/UD60x18.sol";
   import { SD59x18, UNIT as ONE_sd, ZERO as ZERO_sd, sd } from "@prb/
   ↪ math/SD59x18.sol";
```

```
57 import { IERC20 } from "@voltz-protocol/util-contracts/src/
   ↪ interfaces/IERC20.sol";
```

CVF-56 INFO

- **Category** Bad datatype
- **Source** Market.sol

Recommendation The type for this field should be "IERC20".

Client Comment *See CVF-27.*

```
67 address quoteToken;
```

CVF-57 INFO

- **Category** Suboptimal
- **Source** Market.sol

Description In ERC20 the ”decimals” property is used by UI to render token amounts in a human-readable way. Using this property in smart contracts is discouraged.

Recommendation Consider treating all token amounts as integers.

```
68  uint8 quoteTokenDecimals;
```

```
133 market.quoteTokenDecimals = IERC20(quoteToken).decimals();
```

CVF-58 FIXED

- **Category** Bad datatype
- **Source** Market.sol

Recommendation The value ”18” should be a named constant.

```
143 return sd(DecimalMath.changeDecimals(a, self.quoteTokenDecimals, 18,
    ↪  roundingUp));
```

```
147 return DecimalMath.changeDecimals(a.unwrap(), 18, self.
    ↪ quoteTokenDecimals, roundingUp);
```

```
151 return ud(DecimalMath.changeDecimals(a, self.quoteTokenDecimals, 18,
    ↪  roundingUp));
```

CVF-60 INFO

- **Category** Bad datatype
- **Source** GlobalConfiguration.sol

Recommendation The type for these fields should be more specific.

Client Comment *Settable only by owner, we don’t worry about misconfiguration.*

```
23  address coreProxy;
```

```
27  address exchangeProxy;
```

CVF-61 INFO

- **Category** Bad datatype
- **Source** GlobalConfiguration.sol

Recommendation The return type should be more specific.

Client Comment *Settable only by owner, we don't worry about misconfiguration.*

```
48 function getCoreProxyAddress() internal view returns (address) {
```

```
52 function getOracleManagerAddress() internal view returns (address) {
```

```
56 function getExchangeProxyAddress() internal view returns (address) {
```

CVF-62 INFO

- **Category** Procedural
- **Source** RebateConfiguration.sol

Description We didn't review this file.

```
12 import { UD60x18 } from "@prb/math/UD60x18.sol";
```

CVF-63 FIXED

- **Category** Bad datatype
- **Source** RebateConfiguration.sol

Recommendation The storage slot address calculated here should be a named constant.

```
21 bytes32 s = keccak256(abi.encode("xyz.reya.RebateConfiguration"));
```

CVF-64 INFO

- **Category** Procedural
- **Source** TakerFeeConfiguration.sol

Description We didn't review this file.

```
12 import { UD60x18 } from "@prb/math/UD60x18.sol";
```

CVF-65 INFO

- **Category** Procedural
- **Source** ConfigurationModule.sol

Description We didn't review these files.

```
21 import { UD60x18 } from "@prb/math/UD60x18.sol";
   import { OwnableStorage } from "@voltz-protocol/util-contracts/src/
      ↪ storage/OwnableStorage.sol";
```

CVF-67 INFO

- **Category** Procedural
- **Source** FeatureFlagModule.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
19 contract FeatureFlagModule is BaseFeatureFlagModule { }
```

CVF-68 INFO

- **Category** Procedural
- **Source** OwnerUpgradeModule.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
14 contract OwnerUpgradeModule is BaseOwnerUpgradeModule { }
```

CVF-69 FIXED

- **Category** Procedural
- **Source** PassivePerpInstrumentModule.sol

Description This version requirement is inconsistent with other files in the same code base. Also, it is not recommended to use version requirements without upper limit for major version, as it is impossible to guarantee compatibility with not yet released major versions.

Recommendation Consider specifying as "^0.8.19".

Client Comment *Made it considtent with every other file (pragma solidity >=0.8.19 <0.9.0;).*

```
8 pragma solidity >=0.8.19;
```

CVF-70 INFO

- **Category** Procedural
- **Source** PassivePerpInstrumentModule.sol

Description We didn't review these files.

```
34 import { IERC165 } from "@voltz-protocol/util-contracts/src/
     ↪ interfaces/IERC165.sol";
```

```
45 import { UD60x18, ZERO as ZERO_ud } from "@prb/math/UD60x18.sol";
   import { SD59x18, ZERO as ZERO_sd } from "@prb/math/SD59x18.sol";
```

CVF-71 INFO

- **Category** Procedural
- **Source** PassivePerpInstrumentModule.sol

Recommendation This should go first to ensure the market does exist.

Client Comment *We could not find any places where there's a need to check the market exists first.*

```
96 Market.Data storage market = Market.exists(marketId);
```

```
292 Market.Data storage market = Market.exists(marketId);
```

CVF-72 FIXED

- **Category** Readability
- **Source** PassivePerpInstrumentModule.sol

Recommendation Should be ”else if”.

227
```
if (liquidationType == LiquidationType.Backstop) {
```

252
```
if (liquidationType == LiquidationType.ADL) {
```

CVF-73 INFO

- **Category** Procedural
- **Source** DataTypes.sol

Description We didn’t review these files.

10
```
import { UD60x18 } from "@prb/math/UD60x18.sol";
import { SD59x18 } from "@prb/math/SD59x18.sol";
```

CVF-74 FIXED

- **Category** Documentation
- **Source** DataTypes.sol

Description These two comments are identical, while belonging to different fields.

Recommendation Consider adding more details to distinguish them.

66
```
* @dev  ).
* Note, this is different from the pSlippage calculated for the
   ↪ passive pool when it's␣exposure␣is␣affected
*␣by␣a␣trade␣or␣a␣liquidation
␣␣␣␣␣␣
```

72
```
* @dev Configurable parameter used in the percentual price slippage
   ↪ computation (used for liquidations).
* Note, this is different from the pSlippage calculated for the
   ↪ passive pool when it's␣exposure␣is␣affected
*␣by␣a␣trade␣or␣a␣liquidation
␣␣␣␣␣␣
```

CVF-75 INFO

- **Category** Procedural
- **Source** Errors.sol

Description We didn't review these files.

```
11 import { UD60x18 } from "@prb/math/UD60x18.sol";
   import { SD59x18 } from "@prb/math/SD59x18.sol";
```

CVF-76 FIXED

- **Category** Procedural
- **Source** Errors.sol

Recommendation These errors should be made more useful by adding certain parameters into them.

Client Comment *Agreed for the order size errors. LiquidationAgainstPassivePool does not require more information and adding more information to InvalidOrderCounterparties increases complexity.*

```
34 error LiquidationAgainstPassivePool();
```

```
57 error SmallOrderSize();
```

```
63 error DustyOrderSize();
```

```
69 error InvalidOrderCounterparties();
```

CVF-77 INFO

- **Category** Procedural
- **Source** Prices.sol

Description We didn't review these files.

```
11 import { UD60x18, ZERO as ZERO_ud } from "@prb/math/UD60x18.sol";
   import { SD59x18, ZERO as ZERO_sd } from "@prb/math/SD59x18.sol";
```

CVF-78 INFO

- **Category** Readability
- **Source** Prices.sol

Recommendation Should be ”else return”.

Client Comment *We prefer to avoid this pattern.*

```
26 return roundedDown.add(priceSpacing);
```

CVF-79 INFO

- **Category** Readability
- **Source** Prices.sol

Recommendation Should be ”else return”.

Client Comment *We prefer to avoid this pattern.*

```
60 return updatedPrice.intoUD60x18();
```

CVF-80 INFO

- **Category** Procedural
- **Source** PerpPositionSupport.sol

Description We didn’t review these files.

```
14 import { UD60x18, UNIT as ONE_ud } from "@prb/math/UD60x18.sol";
   import { SD59x18, ZERO as ZERO_sd } from "@prb/math/SD59x18.sol";
```

CVF-81 INFO

- **Category** Readability
- **Source** PerpPositionSupport.sol

Description The rest of the function looks like it is always executed, while actually it is executed only in case the condition above is false.

Recommendation Consider putting the rest of the function into an explicit ”else” block.

Client Comment *We prefer avoiding the ”if else” nesting and consider this pattern to be cleaner.*

```
75 }
```

```
95 }
```

```
113 }
```

CVF-82 INFO

- **Category** Readability
- **Source** PerpPositionSupport.sol

Description The rest of the function looks like it is always executed, while actually it is executed only in case the condition above is false.

Recommendation Consider putting the rest of the function into an explicit ”else” block.

Client Comment *We prefer avoiding the ”if else” nesting and consider this pattern to be cleaner.*

```
145 }
```

```
164 }
```

CVF-83 INFO

- **Category** Procedural
- **Source** MatchOrders.sol

Description We didn't review these files.

```
13 import { UD60x18, ZERO, UNIT } from "@prb/math/UD60x18.sol";
   import { SD59x18, ZERO as ZERO_sd } from "@prb/math/SD59x18.sol";

17 import { SignedMath } from "oz/utils/math/SignedMath.sol";
```

CVF-84 FIXED

- **Category** Unclear behavior
- **Source** MatchOrders.sol

Description These values should be be calculated in case "orderBase" is zero.

```
23 SD59x18 basePostOrder = netBase.add(orderBase);
   UD60x18 absBasePostOrder = basePostOrder.abs().intoUD60x18();
   UD60x18 absOrderBase = orderBase.abs().intoUD60x18();
```

CVF-85 FIXED

- **Category** Suboptimal
- **Source** MatchOrders.sol

Recommendation This could be simplified as "if (orderBase.eq(ZERO_sd))" and placed earlier.

```
27 // check the incoming order is non-zero
   if (absOrderBase.eq(ZERO)) {
```

CVF-86 FIXED

- **Category** Procedural
- **Source** MatchOrders.sol

Recommendation These checks should be done before validating order size.

66
```
if (counterpartyAccountIds.length != 1) {
```

70
```
if (counterpartyAccountIds[0] != passivePoolAccountId) {
```

CVF-87 FIXED

- **Category** Suboptimal
- **Source** MatchOrders.sol

Description The expression ”takerFeeDebit - passivePoolCredit” is calculated twice.

Recommendation Consider calculating once and reusing.

114
```
uint256 exchangeFeeCredit = mulUDxUint(exchangeRebateParameter,
    ↪ takerFeeDebit - passivePoolCredit);
uint256 protocolFeeCredit = takerFeeDebit - passivePoolCredit -
    ↪ exchangeFeeCredit;
```

CVF-88 INFO

- **Category** Procedural
- **Source** Liquidations.sol

Description We didn’t review these files.

10
```
import { UD60x18 } from "@prb/math/UD60x18.sol";
```

12
```
import { SD59x18, ZERO as ZERO_sd } from "@prb/math/SD59x18.sol";
```

CVF-89 FIXED

- **Category** Documentation
- **Source** Liquidations.sol

Description The semantics of the returned value is unclear.

Recommendation Consider documenting.

```
27 returns (UD60x18)
```

CVF-90 INFO

- **Category** Procedural
- **Source** QuadraticEquation.sol

Description We didn't review these files.

```
12 import { SD59x18, ZERO as ZERO_sd, convert as convert_sd } from "
   ↪ @prb/math/SD59x18.sol";
```

CVF-91 INFO

- **Category** Suboptimal
- **Source** QuadraticEquation.sol

Recommendation The code could be simplified by dividing all three coefficients by 2a.

Client Comment *Our current understanding is the suggestion will only remove the '.div(convert_sd(2).mul(a))' and 'convert_sd(4).mul(a)' but will make the code less readable. From this, gas optimisation is not justified but please highlight if otherwise.*

```
17 function solveQuadraticEquation(SD59x18 a, SD59x18 b, SD59x18 c)
   ↪ pure returns (SD59x18 x1, SD59x18 x2) {
```

CVF-92 FIXED

- **Category** Procedural
- **Source** QuadraticEquation.sol

Recommendation The value ”convert_sd(4)” should be precomputed.

Client Comment *Added as constrant.*

```
22 SD59x18 delta = b.mul(b).sub(convert_sd(4).mul(a).mul(c));
```

CVF-93 FIXED

- **Category** Procedural
- **Source** QuadraticEquation.sol

Recommendation The value ”convert_sd(2)” should be precomputed.

Client Comment *Added as constrant.*

```
30 x1 = flip(b).sub(rootDelta).div(convert_sd(2).mul(a));
   x2 = flip(b).add(rootDelta).div(convert_sd(2).mul(a));
```

CVF-94 FIXED

- **Category** Suboptimal
- **Source** QuadraticEquation.sol

Description The expression ”convert_sd(2).mul(a)” is calculated twice.

Recommendation Consider calculating once and reusing.

```
30 x1 = flip(b).sub(rootDelta).div(convert_sd(2).mul(a));
   x2 = flip(b).add(rootDelta).div(convert_sd(2).mul(a));
```

CVF-95 INFO

- **Category** Procedural
- **Source** MaxExposure.sol

Description We didn’t review these files.

```
13 import { SD59x18, ZERO as ZERO_sd } from "@prb/math/SD59x18.sol";
   import { UD60x18 } from "@prb/math/UD60x18.sol";
```

CVF-96 INFO

- **Category** Procedural
- **Source** FundingRate.sol

Description We didn't review these files.

```
10 import { UD60x18 } from "@prb/math/UD60x18.sol";
   import { SD59x18, convert as convert_sd } from "@prb/math/SD59x18.
      ↪ sol";
```

CVF-97 FIXED

- **Category** Procedural
- **Source** FundingRate.sol

Recommendation The value "convert_sd(2)" should be precomputed.

Client Comment *Added a constant.*

```
61 fundingValueDelta = fundingValueDelta.add(fundingRateDelta.mul(
      ↪ priceSD).mul(periodsSD).div(convert_sd(2)));
```

CVF-98 FIXED

- **Category** Suboptimal
- **Source** FundingRate.sol

Description Both addends here are multiplied by "priceSD" and "periodsSD".

Recommendation Consider adding before multiplication.

```
61 fundingValueDelta = fundingValueDelta.add(fundingRateDelta.mul(
      ↪ priceSD).mul(periodsSD).div(convert_sd(2)));
```

CVF-99 FIXED

- **Category** Readability
- **Source** Constants.sol

Recommendation This value could be rendered as "1 days" in Solidity.

```
10 uint256 constant ONE_DAY_IN_SECONDS = 86_400;
```

CVF-100 INFO

- **Category** Procedural
- **Source** Events.sol

Description We didn't review these files.

```
13 import { MatchOrderFees, LiquidationType } from "@voltz-protocol/
   ↪ core/src/libraries/DataTypes.sol";

15 import { UD60x18 } from "@prb/math/UD60x18.sol";
   import { SD59x18 } from "@prb/math/SD59x18.sol";
```

CVF-101 INFO

- **Category** Bad naming
- **Source** Events.sol

Recommendation Events are usually named via nouns, such as "GlobalConfiguration" or "MarketData".

```
24 event GlobalConfigurationUpdated(GlobalConfiguration.Data
   ↪ globalConfig, uint256 blockTimestamp);

31 event MarketDataUpdated(Market.Data marketData, uint256
   ↪ blockTimestamp);

39 event MarketConfigurationUpdated(uint128 marketId,
   ↪ MarketConfigurationData marketConfig, uint256 blockTimestamp);

48 event PositionDataUpdated(uint128 marketId, uint128 accountId,
   ↪ PerpPosition positionData, uint256 blockTimestamp);

94 event ExchangeRebateUpdated(uint128 exchangeId, UD60x18 rebate,
   ↪ uint256 blockTimestamp);

102 event PoolRebateUpdated(uint128 poolId, UD60x18 rebate, uint256
   ↪ blockTimestamp);

111 event TierFeeUpdated(uint128 marketId, uint256 tierId, UD60x18
   ↪ feeParameter, uint256 blockTimestamp);

120 event AccountTierUpdated(uint128 marketId, uint128 accountId,
   ↪ uint256 tierId, uint256 blockTimestamp);
```

CVF-103 INFO

- **Category** Procedural
- **Source** Permissions.sol

Description This library consists only of constants.

Recommendation Consider moving the constants to the top level and removing the library.

Client Comment *Consistent with Core package & keeps code clean.*

```
10 library Permissions {
```

CVF-104 INFO

- **Category** Procedural
- **Source** IConfigurationModule.sol

Description We didn't review this file.

```
12 import { UD60x18 } from "@prb/math/UD60x18.sol";
```

CVF-105 FIXED

- **Category** Procedural
- **Source** IPassivePerpInformationModule.sol

Recommendation These two imports should be merged.

```
10 import { PerpPosition } from "../libraries/DataTypes.sol";
```

```
13 import { FundingAndADLTrackers, PriceData } from "../libraries/
   ↪ DataTypes.sol";
```

CVF-106 INFO

- **Category** Procedural
- **Source** IPassivePerpInformationModule.sol

Description We didn't review these files.

```
11 import { UD60x18 } from "@prb/math/UD60x18.sol";
   import { SD59x18 } from "@prb/math/SD59x18.sol";
```

CVF-107 INFO

- **Category** Procedural
- **Source** PassivePerpProxy.sol

Description We didn't review this file.

```
10 import "@voltz-protocol/util-contracts/src/proxy/UUPSProxyWithOwner.
   ↪ sol";
```

CVF-108 INFO

- **Category** Bad datatype
- **Source** PassivePerpProxy.sol

Recommendation The type for this argument should be more specific.

```
18 address firstImplementation,
```

CVF-109 INFO

- **Category** Procedural
- **Source** PassivePerpProxy.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
22 { }
```

ABDK
Consulting

About us

Established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function.

The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting