Report

v. 1.0

Customer

Reya Labs

Smart Contract Audit

Reya Network. Part I

2nd April 2024

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	02.04.24	A. Zveryanskaya	Initial Draft
0.2	02.04.24	A. Zveryanskaya	Minor revision
1.0	02.04.24	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 Original Code related to the Core of the Reya Network. Corresponding fixes were provided in the Part III of the Audit.

Files:

core/		
CoreProxy.sol		
core/libraries/		
DataTypes.sol	Errors.sol	Events.sol
FeatureFlagSupport.sol	LiquidationBid PriorityQueue.sol	Liquidation BidQueues.sol
PriceHelpers.sol		
core/libraries/actions/		
EditCollateral.sol	CreateAccount.sol	EditCollateral.sol
core/libraries/account/		
AccountActiveMarket.sol	AccountAuto Exchange.sol	AccountBackstopADL.sol
AccountChecks.sol	AccountExposure.sol	AccountLiquidation.sol
core/interfaces/		
IAccountModule.sol	IAccountToken Module.sol	IAutoExchange ConfigurationModule.sol
IAutoExchange Module.sol	ICollateralAdapter.sol	ICollateral Module.sol
ICollateralPool Module.sol	IExchangeManager Module.sol	IExecutionModule.sol
IInstrumentModule.sol	IInstrument RegistrarModule.sol	IInsuranceFund ConfigurationModule.sol
IProtocolConfiguration Module.sol	IRiskConfiguration Module.sol	

core/interfaces/external/		
ICollateral.sol	IInstrument.sol	ILiquidationHook.sol
INFTPass.sol	IStEth.sol	
core/interfaces/liquidation/		
IBackstopLiquidation Module.sol	ICommonLiquidation Module.sol	IDutchLiquidation Module.sol
IRankedExecute LiquidationModule.sol	IRankedSubmit LiquidationModule.sol	
core/modules/Adapters/		
AaveCollateral Adapter.sol	LidoCollateralAdapter.sol	
core/modules/liquidation/		
BackstopLiquidation Module.sol	CommonLiquidation Module.sol	DutchLiquidation Module.sol
RankedExecute Liquidation Module.sol	RankedSubmitLiquida-tion Module.sol	
core/modules/		
AccountModule.sol	AccountTokenModule.sol	AssociatedSystems Module.sol
AutoExchange ConfigurationModule.sol	AutoExchange Module.sol	CollateralModule.sol
CollateralPoolModule.sol	ExchangeManager Module.sol	ExecutionModule.sol
FeatureFlagModule.sol	InstrumentModule.sol	InstrumentRegistrar Module.sol
InsuranceFund ConfigurationModule.sol	OwnerUpgrade Module.sol	ProtocolConfiguration Module.sol
RiskConfigurationMod-ule.sol		
core/storage/		
Account.sol	AccountCollateral.sol	AccountRBAC.sol
AutoExchange Configuration.sol	BackstopLP Configuration.sol	Collateral Configuration.sol
CollateralPool.sol	Exchange.sol	GlobalCollateral Configuration.sol
IdStore.sol	InstrumentRegistrar.sol	InsuranceFund Configuration.sol
LimitConfiguration.sol	Liquidation Configuration.sol	Market.sol
Protocol Configuration.sol	RiskMatrix.sol	RiskMultipliers Configuration.sol
Signature.sol		

oracle-manager/interfaces/			
	INodeModule.sol		
oracle-manager/interfaces/external/			
	IAggregator V3Interface.sol	IExternalNode.sol	
oracle-manager/modules/			
	NodeModule.sol	OwnerUpgrade Module.sol	
oracle-manager/nodes/			
	ChainlinkNode.sol	ConstantNode.sol	ExternalNode.sol
oracle-manager/			
	OracleManagerProxy.sol		
oracle-manager/storage/			
	NodeDefinition.sol	NodeOutput.sol	
utils/contracts/helpers/			
	PrbMathHelper.sol		
utils/modules/modules/			
	FeatureFlagModule.sol		
utils/modules/storage/			
	FeatureFlag.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 1 critical, 26 major, and a few less important issues. All identified Critical and Major issues have been fixed.

Issues	Info	Fixed
Severity		
Critical	0	1
Major	0	26
Moderate	23	5

Fixed 32 out of 55 issues

6 Critical Issues

CVF-1 FIXED

- **Category** Flaw
- **Source** ExecutionModule.sol

Description The signed message doesn't include the commands nor the caller address. This allows a malicious user to intercept a transaction and modify commands arbitrary.

Recommendation Consider including the commands, or the caller, or both into the signed message.

Client Comment *We included both the commands and the caller.*

```
86 keccak256(abi.encode(EXECUTE_TYPYHASH, accountId, incrementedNonce,
     ↪ sig.deadline))
```

7 Major Issues

CVF-2 FIXED

- **Category** Suboptimal
- **Source** FeatureFlag.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.

```
23 bytes32 s = keccak256(abi.encode("xyz.reya.FeatureFlag", featureName
   ↪ ));
```

CVF-4 FIXED

- **Category** Suboptimal
- **Source** FeatureFlagModule.sol

Recommendation This could be simplified as: flag.deniers = deniers;

```
89 for (uint256 i = 0; i < deniers.length; i++) {
90     if (i >= storageLen) {
           flag.deniers.push(deniers[i]);
       } else {
           flag.deniers[i] = deniers[i];
       }
   }
```

CVF-5 FIXED

- **Category** Suboptimal
- **Source** FeatureFlagModule.sol

Recommendation This could be simplified as: return flag.deniers;

```
105 address[] memory addrs = new address[](flag.deniers.length);
    for (uint256 i = 0; i < addrs.length; i++) {
        addrs[i] = flag.deniers[i];
    }

110 return addrs;
```

CVF-7 FIXED

- **Category** Suboptimal
- **Source** PrbMathHelper.sol

Recommendation This could be optimized as: SD59×18.wrap(a).div(SD59×18.wrap(b)) and UD60×18.wrap(a).div(UD60×18.wrap(b)) Such optimization would also address possible phantom overflow.

```
79 return convert_sd(a).div(convert_sd(b));
```

```
83 return convert_ud(a).div(convert_ud(b));
```

CVF-8 FIXED

- **Category** Suboptimal
- **Source** NodeDefinition.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.

```
37 bytes32 s = keccak256(abi.encode("xyz.reya.oracle-manager.Node", id)
   ↪ );
```

CVF-9 FIXED

- **Category** Suboptimal
- **Source** AccountCollateral.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.

```
58 bytes32 s = keccak256(abi.encode("xyz.reya.AccountCollateral",
   ↪ accountId));
```

CVF-11 FIXED

- **Category** Suboptimal
- **Source** CollateralConfiguration.sol

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

Recommendation Consider using “abi.encodePacked” instead. Also, consider using the hash of a string.

```
58 bytes32 s = keccak256(abi.encode("xyz.reya.CollateralConfiguration",
   ↪  collateralPoolId, collateral));
```

CVF-12 FIXED

- **Category** Suboptimal
- **Source** RiskMultipliersConfiguration.sol

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

Recommendation Consider using the “abi.encodePacked” instead. Also consider using the hash of a string.

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

CVF-13 FIXED

- **Category** Suboptimal
- **Source** LiquidationConfiguration.sol

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

Recommendation Consider using the “abi.encodePacked” function instead. Also, consider using the hash of a string.

```
25 bytes32 s = keccak256(abi.encode("xyz.reya.LiquidationConfiguration"
   ↪ , id));
```

CVF-14 FIXED

- **Category** Suboptimal
- **Source** AutoExchangeConfiguration.sol

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

Recommendation Consider using the “abi.encodePacked” function instead. Also, consider using the hash of a string.

```
25 bytes32 s = keccak256(abi.encode("xyz.reya.AutoExchangeConfiguration
   ↪ ", id));
```

CVF-15 FIXED

- **Category** Suboptimal
- **Source** LimitConfiguration.sol

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

Recommendation Consider using the “abi.encodePacked” function instead. Also, consider using the hash of a string.

```
22 bytes32 s = keccak256(abi.encode("xyz.voltz.LimitConfiguration", id)
   ↪ );
```

CVF-16 FIXED

- **Category** Suboptimal
- **Source** BackstopLPConfiguration.sol

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

Recommendation Consider using the “abi.encodePacked” function instead. Also, consider using the hash of a string.

```
30 bytes32 s = keccak256(abi.encode("xyz.reya.BackstopLPConfiguration",
   ↪  id));
```

CVF-17 FIXED

- **Category** Suboptimal
- **Source** InsuranceFundConfiguration.sol

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

Recommendation Consider using the “abi.encodePacked” function instead. Also, consider using the hash of a string.

```
26 bytes32 s = keccak256(abi.encode("xyz.reya.
   ↪ InsuranceFundConfiguration", id));
```

CVF-20 FIXED

- **Category** Suboptimal
- **Source** CollateralPool.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.

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

CVF-21 FIXED

- **Category** Suboptimal
- **Source** AccountRBAC.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.AccountRBAC", accountId))
   ↪ ;
```

CVF-23 FIXED

- **Category** Suboptimal
- **Source** Account.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.

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

CVF-24 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.

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

CVF-25 FIXED

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

Description The “caller” argument name is confusing, as it is not guaranteed to be the caller.

Recommendation Consider renaming or using “msg.sender” instead of the argument.

Client Comment*Fixed by using msg.sender instead.*

```
126 function onlyMarketAddress(uint128 marketId, address caller)
      ↪ internal view {
```

CVF-26 FIXED

- **Category** Suboptimal
- **Source** GlobalCollateralConfiguration.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.

```
103 bytes32 s = keccak256(abi.encode("xyz.reya.
    ↪ GlobalCollateralConfiguration", collateral));
```

CVF-27 FIXED

- **Category** Suboptimal
- **Source** RiskMatrix.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.

```
45 bytes32 s = keccak256(abi.encode("xyz.reya.CollateralPoolRiskMatrix"
   ↪ , collateralPoolId));
```

CVF-33 FIXED

- **Category** Suboptimal
- **Source** IdStore.sol

Description ABI encoding of a string is more complicated than the string itself, as it contains the string length.

Recommendation Consider just casting string to "bytes" instead of ABI encoding it.

Client Comment*Fixed by replacing with 'keccak256(bytes("..."));'.*

```
16 bytes32 private constant _SLOT_ID_STORE = keccak256(abi.encode("xyz.
      ↪ reya.IdStore"));
```

```
18 bytes32 private constant ACCOUNT_ID = keccak256(abi.encode("
      ↪ AccountId"));
   bytes32 private constant MARKET_ID = keccak256(abi.encode("MarketId"
      ↪ ));
20 bytes32 private constant EXCHANGE_ID = keccak256(abi.encode("
      ↪ ExchangeId"));
   bytes32 private constant RISK_BLOCK_ID = keccak256(abi.encode("
      ↪ RiskBlockId"));
```

CVF-34 FIXED

- **Category** Suboptimal
- **Source** IdStore.sol

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

Recommendation Consider using the written value.

```
43 id = idStore.lastIds[idType];
```

CVF-35 FIXED

- **Category** Suboptimal
- **Source** InstrumentRegistrar.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.InstrumentRegistrar"));
```

CVF-40 FIXED

- **Category** Documentation
- **Source** AccountChecks.sol

Description This comment gives no clue, why "<=" is fine here.

Recommendation Consider elaborating more on this.

Client Comment*Removed this comment. It is not suggestive. <= or <, any should be fine because we are checking a buffer range.*

```
91 // it's fine to use <= here
```

CVF-42 FIXED

- **Category** Suboptimal
- **Source** AccountBackstopADL.sol

Description The expression "AccountCollateral.getPool(self.id)" is calculated several times.

Recommendation Consider calculating once and reusing.

```
68 bool isBackstopLpAccount = AccountCollateral.getPool(self.id).
   ↪ hasBackstopLPAccount();
```

```
74     Account.Data storage backstopLpAccount = AccountCollateral.
          ↪ getPool(self.id).getBackstopLPAccount();
```

```
93         Account.Data storage backstopLpAccount = AccountCollateral.
              ↪ getPool(self.id).getBackstopLPAccount();
```

CVF-49 FIXED

- **Category** Documentation
- **Source** IInstrument.sol

Description The semantics and the format of the "output" field is unclear.

Recommendation Consider documenting.

```
70 returns (bytes memory output, MatchOrderFees memory matchOrderFees);
```

```
146 returns (bytes memory output, int256 cashflowAmount);
```

8 Moderate Issues

CVF-3 INFO

- **Category** Suboptimal
- **Source** FeatureFlag.sol

Recommendation This function could be made much more efficient by requiring the deniers to be stored in sorted order.

Client Comment*Making this more efficient does not bring a lot of gas benefits as this function is only used when permssions are changed, which happens rarely.*

```
61 function isDenier(Data storage self, address possibleDenier)
     ↪ internal view returns (bool) {
```

CVF-28 INFO

- **Category** Suboptimal
- **Source** RiskMatrix.sol

Description Here a risk matrix is stored one value per storage slot, which could be very expensive.

Recommendation Consider packing several values into a single slot using a floating point or a narrower fixed point number format. For example, as absolute values of risk matrix elements may not exceed one, there is no need to store the integer parts of them.

Client Comment*Adds complexity.*

```
60 riskBlockMatrix.blockMatrix[blockId] = values;
```

CVF-29 INFO

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

Description As a risk matrix is guaranteed to be symmetric, only half of it actually ought to be stored.

Client Comment*This introduces some complexity for the read operations which prioratize above the write operation in terms of gas.*

```
60 riskBlockMatrix.blockMatrix[blockId] = values;
```

CVF-30 INFO

- **Category** Suboptimal
- **Source** RiskMatrix.sol

Description Each matrix row is stored here as a separate array, which means that length is stored for each row. This is redundant, as lengths of all rows are equal, and are the same as the number for rows.

Recommendation There is no need to store row lengths at all.

Client Comment*This introduces some complexity for the read operations which prioratize abive the write operation in terms of gas.*

```
60 riskBlockMatrix.blockMatrix[blockId] = values;
```

CVF-31 INFO

- **Category** Suboptimal
- **Source** RiskMatrix.sol

Description This shouldn't be checked for the case i==j. Also, each pair is checked for equality twice.

Recommendation Consider refactoring to avoid redundant checks.

Client Comment*This might add extra complexity.*

```
143 // check for symmetry
    if (!values[i][j].eq(values[j][i])) {
```

CVF-32 INFO

- **Category** Suboptimal
- **Source** Signature.sol

Description Calculating the domain separator every time is suboptimal.

Recommendation Common optimization is to calculate it once in the constructor and store in an immutable variable along with chain ID. Then, if chain ID didn't change since constructor invocation, use the stored domain separator. Otherwise calculate it.

Client Comment*With the proxy structure we use, we prefer not to change anything in the constructor.*

```
76 return keccak256(
       abi.encode(
           EIP712_DOMAIN_TYPEHASH, keccak256(bytes("Reya")),
              ↪ EIP712_REVISION_HASH, block.chainid, address(this)
       )
80 );
```

CVF-36 INFO

- **Category** Suboptimal
- **Source** Exchange.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.

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

CVF-37 INFO

- **Category** Suboptimal
- **Source** EditCollateral.sol

Description This check is redundant. It will anyway be performed inside the “transferFrom” call. Also, this check doesn’t guarantee successful transfer.

Recommendation Consider removing this check.

Client Comment*This check is added because each ERC20 reverts in their own way and affects how the off-chain behaves.*

```
54 uint256 allowance = IERC20(collateral).allowance(depositFrom, self);
   if (allowance < collateralAmount) {
```

CVF-38 INFO

- **Category** Suboptimal
- **Source** EditCollateral.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.

```
145 return keccak256(abi.encode("backstopLpWithdrawTimer", accountId));
```

CVF-41 INFO

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

Description There is no range check for this argument.

Recommendation Consider adding an appropriate check.

Client Comment*‘validateBackstopLiquidation‘ validates if this number is < 1e18. There is no place in the code where this parameter is used before validation. Yes, there are util functions (internal ones) that use it but the assumption is that validation happened already.*

```
49 UD60x18 backstopPercentage
```

CVF-43 INFO

- **Category** Suboptimal
- **Source** AccountBackstopADL.sol

Recommendation This formula could be rewritten as absUPnLs[i] * (-existingFunds / absUPnLSum), where the part in brackets could be computed once before the loop.

Client Comment *This change requires checking if absUPnLSum is zero. It also involves changing tests (due to small precision changes).*

```
236 supportedUPnL[i] = mulUDxInt(divUintUint(absUPnLs[i], absUPnLSum), -
    ↪ existingFunds);
```

CVF-44 INFO

- **Category** Suboptimal
- **Source** LiquidationBidPriorityQueue.sol

Description The value "_score(self, i)" is calculated on every iteration, while this value stands the same during the loop.

Recommendation Consider calculating this value once before the loop and reusing.

Client Comment *"i" changes and it implicitly changes the value of this score.*

```
67 if (_score(self, i) > _score(self, j)) {
```

CVF-45 INFO

- **Category** Suboptimal
- **Source** LiquidationBidPriorityQueue.sol

Description The value "_score(self, j)" was already calculated a few lines above, either as "_score(self, j)" or as "_score(self, j + 1)".

Recommendation Consider reusing the already calculated value.

Client Comment *This introduces complexity.*

```
67 if (_score(self, i) > _score(self, j)) {
```

CVF-46 INFO

- **Category** Suboptimal
- **Source** LiquidationBidPriorityQueue.sol

Description The score and the index of the new element are read from the storage on each iteration, while these values are actually available on stack as an argument as a local variable.

Recommendation Consider using values from stack, rather than from the storage.

Client Comment *This introduces complexity.*

```
96 while (i > 1 && _score(self, i / 2) < _score(self, i)) {
       (self.indices[i / 2], self.indices[i]) = (self.indices[i], self.
          ↪ indices[i / 2]);
```

CVF-47 INFO

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

Recommendation The “blockTimestamp” parameters are redundant as each event is anyway bound to a block, whose timestamp could be easily obtained.

Client Comment*The off-chain infrastructure uses this parameter. Sometimes, only the block-number is exposed and requires a second RPC call to obtain the timestamp. Please let us know if this is also a gas optimisation issue, we might reconsider.*

```
37  event AccountOwnerUpdated(uint128 indexed accountId, address indexed
        ↪  newOwner, uint256 blockTimestamp);

47      uint128 indexed accountId, bytes32 indexed permission, address
            ↪ indexed user, uint256 blockTimestamp

58      uint128 indexed accountId, bytes32 indexed permission, address
            ↪ indexed user, uint256 blockTimestamp

69      uint128 indexed accountId, address indexed collateral, int256
            ↪ sharesDelta, uint256 blockTimestamp

78  event AccountNewActiveMarket(uint128 indexed accountId, uint128
        ↪ marketId, uint256 blockTimestamp);

85      uint256 blockTimestamp

93      uint256 blockTimestamp

102     uint128 indexed collateralPoolId, address collateralAddress,
            ↪ CollateralConfig newConfig, uint256 blockTimestamp

114     uint256 blockTimestamp

117 event CollateralPoolCreation(uint128 indexed collateralPoolId,
        ↪ uint128 reservedAccountId, uint256 blockTimestamp);

119 event CollateralPoolOwnerUpdated(uint128 indexed collateralPoolId,
        ↪ address newOwner, uint256 blockTimestamp);

122     uint128 indexed collateralPoolId, address indexed collateral,
            ↪ int256 sharesDelta, uint256 blockTimestamp
```

(126, 129, 132, 136, 140, 144, 147, 149, 152, 158, 167, 174, 177, 185, 193, 207, 210, 220, 239, 258)

CVF-48 INFO

- **Category** Suboptimal
- **Source** DataTypes.sol

Recommendation A bit mask of permissions would be much more efficient.

Client Comment*A bit mask would introduce complexity and getAccountPermissions(uint128 accountId) is not frequently used to justify.*

```
57  * @dev The array of permissions given to the associated address.
```

```
59 bytes32[] permissions;
```

CVF-50 INFO

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

Description This allows flags :”allowAll” and ”denyAll” to be set simultaneously.

Recommendation Consider forbidding such situation.

Client Comment*The flag denyAll has prioroty over allowAll. Introducing blocks for such situation would add complexity.*

```
41 flag.denyAll = denyAll;
```

CVF-51 FIXED

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

Description Despite the name, this function calculates not the TWAP, but rather the average of all prices available within the given time interval.

Recommendation Proper TWAP calculation should use not only the prices, but also their timestamps.

```
39 function getTwapPrice(
```

CVF-52 FIXED

- **Category** Flaw
- **Source** ChainlinkNode.sol

Recommendation This check should be surrounded with a ”try/catch” block, to return false on error, rather then revert transaction.

Client Comment*Fixed by adding try/catch block.*

```
88 // Must return latestRoundData without error
   chainlink.latestRoundData();
```

CVF-53 INFO

- **Category** Procedural
- **Source** AccountCollateral.sol

Description Using a business-level field for a low-level purpose of detecting, whether pool does exist is a bad practice. In the future, zero value for the “firstMarketId” field could become valid.

Recommendation Consider using a separate low-level flag instead.

Client Comment*As a general practice with ID’s, id = 0 means not registered across the protocol. We use the IdStore which assigns incremental ids such that no assignament is manual.*

```
66 return accountCollateral.firstMarketId != 0;
```

CVF-54 INFO

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

Description This code is executed even if “sharesDelta” is negative. Also, there is no logic to remove a collateral from “activeCollateral” set once the number of its shares drops to zero.

Client Comment*Lacking the removal logic was a decision we took because of the complexity it would add. The assumption is that the number of possible collaterals will not bulk this array.*

```
329 if (!self.activeCollaterals.contains(collateral)) {
330     self.activeCollaterals.add(collateral);
```

CVF-55 INFO

- **Category** Suboptimal
- **Source** CollateralPool.sol

Description The “existingCollaterals” field is incremented even if the new parent collateral was already supported.

Recommendation Consider properly handling such a case.

Client Comment*This case is handled. existingCollaterals += 1 counts the ‘collateral‘. The 1st if-statement executes an existingCollaterals -= 1 if the callateral was already registered or does nothing if not registered. So the existingCollaterals += 1 operation either restores the initial state if the collateral was registered prior to this function or adds a new collateral.*

```
440 self.supportingCollaterals[newParentCollateral].add(collateral);
    self.existingCollaterals += 1;
```

CVF-56 INFO

- **Category** Suboptimal
- **Source** RiskMatrix.sol

Description This check doesn’t make sense for the case when “i” equals “j”.

Recommendation Consider handling this case separately.

Client Comment*This validation only happens when creating a new matrx which is not a frequent operation. if i = j this check should pass which is expected. skipping this case would introduce code complexity.*

```
143 // check for symmetry
    if (!values[i][j].eq(values[j][i])) {
```

CVF-57 FIXED

- **Category** Flaw
- **Source** AaveCollateralAdapter.sol

Description Multiplying assets or shares by totalShares makes usage of the “mulDiv” function worthless, as multiplication overflow would anyway cause the transaction to be reverted.

Recommendation Consider using the “mulmod” function to check whether precision was lost.

Client Comment*Fixed by replacing with mulmod.*

```
35 shares = mulDiv(assets, totalShares, totalSupply);
   precisionLoss = (assets * totalShares) - (shares * totalSupply);
```

```
48 assets = mulDiv(shares, totalSupply, totalShares);
   precisionLoss = (shares * totalSupply) - (assets * totalShares);
```

CVF-58 FIXED

- **Category** Flaw
- **Source** LidoCollateralAdapter.sol

Description Multiplying assets or shares by totalShares makes usage of the “mulDiv” function worthless, as multiplication overflow would anyway cause the transaction to be reverted.

Recommendation Consider using the “mulmod” function to check whether precision was lost.

Client Comment*Fixed by replacing with mulmod.*

```
35 shares = mulDiv(assets, totalShares, totalSupply);
   precisionLoss = (shares * totalSupply) - (assets * totalShares);
```

```
48 assets = mulDiv(shares, totalSupply, totalShares);
   precisionLoss = (assets * totalShares) - (shares * totalSupply);
```

CVF-59 INFO

- **Category** Procedural
- **Source** AutoExchangeModule.sol

Description A comment inside the ”AutoExchange.triggerAutoExchange” function warn about possible reentrancy attacks, while here the function is called without any reentrancy protection.

Recommendation Consider addressing this property or removing the comment in case it is not relevant.

Client Comment*The only external call is to the token adapter (when conversiong from assets to shares). This does not represent an issue because the protocol owner is the only one able to set the adapters. But worth noting for future adapter development that only trusted contracts should be called.*

```
32 return AutoExchange.triggerAutoExchange(input);
```

CVF-60 INFO

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

Description It is unclear how this concern is supposed to be addressed. This code is called from a function that doesn’t have any reentrancy protection.

Recommendation Consider implementing batch balance update functionality in the “AccountCollateral” library, that would guarantee no external calls during an update.

Client Comment*Same as CVF-59.*

```
87 /// @dev reentrancy risk while account balance is lower, e.g.
      ↪ execute liquidation
   /// through the collateral adapter external call
```

CVF-61 FIXED

- **Category** Procedural
- **Source** EditCollateral.sol

Recommendation This check should be performed before sending tokens out. Otherwise, user may handle incoming token transfer and execute some logic, that will resupply account balance with other tokens. For example, user may exchange withdrawn token to other tokens and deposit them, effectively implementing flash loan functionality.

Client Comment*By just moving the transfer collateral.safeTransfer(msg.sender, collateralAmount); at the end of the function we can avoid such issue.*

105
```
CollateralInfo memory quoteCollateralInfo = account.
    ↪ getCollateralInfo(collateral);
AccountChecks.checkPositiveRealBalance(account.id,
    ↪ quoteCollateralInfo);
```

9 Recommendations

CVF-62 INFO

- **Category** Procedural
- **Source** FeatureFlag.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. Relevant for files: FeatureFlagModule.sol, PrbMathHelper.sol, NodeOutput.sol, NodeDefinition.sol, ExternalNode.sol, ChainlinkNode.sol, ConstantNode.sol, NodeModule.sol, OwnerUpgradeModule.sol, IAggregatorV3Interface.sol, IExternalNode.sol, INodeModule.sol, OracleManagerProxy.sol, AccountCollateral.sol, ProtocolConfiguration.sol, CollateralConfiguration.sol, RiskMultipliersConfiguration.sol, LiquidationConfiguration.sol, AutoExchangeConfiguration.sol, LimitConfiguration.sol, BackstopLPConfiguration.sol, InsuranceFundConfiguration.sol, CollateralPool.sol, AccountRBAC.sol, Account.sol, LiquidationBidQueues.sol, Market.sol,GlobalCollateralConfiguration.sol, RiskMatrix.sol, Signature.sol IdStore.sol, Exchange.sol, BackstopLiquidationModule.sol, CommonLiquidationModule.sol, DutchLiquidationModule.sol, RankedExecuteLiquidationModule.sol, RankedSubmitLiquidationModule.sol, AaveCollateralAdapter.sol, LidoCollateralAdapter.sol, AccountModule.sol,AccountTokenModule.sol, AssociatedSystemsModule.sol, AutoExchangeConfigurationModule.sol, CollateralModule.sol, CollateralPoolModule.sol, ExchangeManagerModule.sol, ExecutionModule.sol, InstrumentModule.sol, InstrumentRegistrarModule.sol, InsuranceFundConfigurationModule.sol, ProtocolConfigurationModule.sol, RiskConfigurationModule.sol,AutoExchangeModule.sol, OwnerUpgradeModule.sol, FeatureFlagModule.sol, AutoExchange.sol, CreateAccount.sol, EditCollateral.sol, AccountActiveMarket.sol, AccountExposure.sol,AccountAutoExchange.sol, AccountChecks.sol, AccountBackstopADL.sol, FeatureFlagSupport.sol, LiquidationBidPriorityQueue.sol, Events.sol, PriceHelpers.sol, DataTypes.sol, Errors.sol, IBackstopLiquidationModule.sol, ICommonLiquidationModule.sol, IDutchLiquidationModule.sol, IRankedExecuteLiquidationModule.sol, IRankedSubmitLiquidationModule.sol, IACollateral.sol, IInstrument.sol, ILiquidationHook.sol, INFTPass.sol, IStEth.sol, IAccountTokenModule.sol, IAutoExchangeConfigurationModule.sol, IAccountModule.sol, IAutoExchangeModule.sol, ICollateralAdapter.sol, ICollateralModule.sol, IExecutionModule.sol, IInsuranceFundConfigurationModule.sol, IInstrumentRegistrarModule.sol, IInstrumentModule.sol, IExchangeManagerModule.sol, ICollateralPoolModule.sol, IProtocolConfigurationModule.sol, IRiskConfigurationModule.sol, CoreProxy.sol.

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

CVF-63 INFO

- **Category** Procedural
- **Source** FeatureFlag.sol

Description We didn't review these files.

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

CVF-64 INFO

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

Description We didn't review these files.

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

CVF-65 FIXED

- **Category** Suboptimal
- **Source** FeatureFlagModule.sol

Description The same flag is loaded twice.

Recommendation Consider loading once and reusing.

```
19 FeatureFlag.Data storage flag = FeatureFlag.load(feature);
```

```
25     FeatureFlag.load(feature).denyAll = false;
```

CVF-66 INFO

- **Category** Suboptimal
- **Source** FeatureFlagModule.sol

Recommendation This check seems redundant, as adding an existing element to a set usually does nothing. Also, the ”add” function should return a boolean value indicating whether the element was actually added or not, which value could be used to conditionally emit an event.

Client Comment *Same SetUtil problem, also controlling an event emit.*

```
55 if (!permissionedAddresses.contains(account)) {
       permissionedAddresses.add(account);
```

CVF-67 FIXED

- **Category** Suboptimal
- **Source** FeatureFlagModule.sol

Description The same flag is loaded three times.

Recommendation Consider loading once and reusing.

```
65 FeatureFlag.Data storage flag = FeatureFlag.load(feature);
```

```
68 SetUtil.AddressSet storage permissionedAddresses = FeatureFlag.load(
       ↪ feature).permissionedAddresses;
```

```
71     FeatureFlag.load(feature).permissionedAddresses.remove(account);
```

CVF-68 INFO

- **Category** Procedural
- **Source** PrbMathHelper.sol

Description We didn’t review these files.

```
3 import {UD60x18, mul as mulUD60x18, div as divUD60x18, intoSD59x18,
      ↪ convert as convert_ud} from "@prb/math/UD60x18.sol";
  import {SD59x18, mul as mulSD59x18, div as divSD59x18, convert as
      ↪ convert_sd} from "@prb/math/SD59x18.sol";
  import {SafeCastU256, SafeCastI256} from "./SafeCast.sol";
```

CVF-69 INFO

- **Category** Procedural
- **Source** NodeOutput.sol

Description This library consists only of type definitions.

Recommendation Consider moving the type definitions into the top level and removing the library.

Client Comment *We prefer having a separate file. Please highlight if this affects gas.*

```
10 library NodeOutput {
```

CVF-70 FIXED

- **Category** Suboptimal
- **Source** NodeDefinition.sol

Recommendation It would be more efficient to use the ”abi.encodePacked” function here, as it doesn’t encode array lengths, which is fine in this particular case.

```
60 return keccak256(abi.encode(nodeDefinition.nodeType, nodeDefinition.
   ↪ parameters, nodeDefinition.parents));
```

CVF-71 INFO

- **Category** Procedural
- **Source** ExternalNode.sol

Description We didn’t review this file.

```
10 import { ERC165Helper } from "@voltz-protocol/util-contracts/src/
   ↪ helpers/ERC165Helper.sol";
```

CVF-72 FIXED

- **Category** Suboptimal
- **Source** ExternalNode.sol

Recommendation This could be simplified as: IExternalNode externalNode = abi.decode (parameters, (IExternalNode));

```
24 IExternalNode externalNode = IExternalNode(abi.decode(parameters, (
      ↪ address)));
```

CVF-74 FIXED

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

Recommendation The type for the ”chainlinkAddr” should be ”IAggregatorV3Interface”.

```
22 error NegativePrice(int256 price, address chainlinkAddr);
```

CVF-75 FIXED

- **Category** Readability
- **Source** ChainlinkNode.sol

Recommendation This could be simplified as: (IAggregatorV3Interface chainlink, uint256 twapTimeInterval) = abi.decode(parameters, (IAggregatorV3Interface, uint256));

```
25 (address chainlinkAddr, uint256 twapTimeInterval) = abi.decode(
      ↪ parameters, (address, uint256));
   IAggregatorV3Interface chainlink = IAggregatorV3Interface(
      ↪ chainlinkAddr);
```

CVF-76 FIXED

- **Category** Readability
- **Source** NodeModule.sol

Recommendation Should be ”else revert”.

```
135 revert UnprocessableNode(nodeId);
```

CVF-78 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-79 INFO

- **Category** Suboptimal
- **Source** IAggregatorV3Interface.sol

Recommendation Consider importing this interface from the Chainlink repository, instead of copying it.

Client Comment *Keeping for better control over the interface.*

```
13 interface IAggregatorV3Interface {
```

CVF-80 INFO

- **Category** Procedural
- **Source** IExternalNode.sol

Description We didn't review this file.

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

CVF-81 INFO

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

Recommendation Events are usually named via nouns, such as "Node".

Client Comment *Acknowledged without action.*

```
39 event NodeRegistered(bytes32 nodeId, NodeDefinition.NodeType
   ↪ nodeType, bytes parameters, bytes32[] parents);
```

CVF-82 INFO

- **Category** Procedural
- **Source** OracleManagerProxy.sol

Description We didn't review this file.

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

CVF-83 INFO

- **Category** Procedural
- **Source** OracleManagerProxy.sol

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

```
22 { }
```

CVF-84 INFO

- **Category** Procedural
- **Source** AccountCollateral.sol

Description We didn't review these files.

```
10 import { SafeCastU256, SafeCastI256 } from "@voltz-protocol/util-
   ↪ contracts/src/helpers/SafeCast.sol";
   import { SetUtil } from "@voltz-protocol/util-contracts/src/helpers/
   ↪ SetUtil.sol";
```

CVF-85 INFO

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

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

Client Comment *See CVF-110.*

```
24 address oracleManagerAddress;
```

CVF-86 INFO

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

Recommendation The type for these fields should be “INFTPass”.

Client Comment *See CVF-110.*

```
29 address accessPassNFTAddress;
```

```
36 address exchangePassNFTAddress;
```

CVF-87 INFO

- **Category** Procedural
- **Source** CollateralConfiguration.sol

Description We didn’t review these files.

```
15 import { INodeModule } from "@voltz-protocol/oracle-manager/src/
   ↪ interfaces/INodeModule.sol";
```

```
23 import { NodeOutput } from "@voltz-protocol/oracle-manager/src/
   ↪ storage/NodeOutput.sol";
   import { SetUtil } from "@voltz-protocol/util-contracts/src/helpers/
   ↪ SetUtil.sol";
   import { Time } from "@voltz-protocol/util-contracts/src/helpers/
   ↪ Time.sol";
```

```
27 import { SafeCastI256 } from "@voltz-protocol/util-contracts/src/
   ↪ helpers/SafeCast.sol";
   import { UD60x18, UNIT, ud } from "@prb/math/UD60x18.sol";
   import { mulUDxUint } from "@voltz-protocol/util-contracts/src/
   ↪ helpers/PrbMathHelper.sol";
```

CVF-88 FIXED

- **Category** Procedural
- **Source** CollateralConfiguration.sol

Recommendation Duplicated import.

```
17 import { GlobalCollateralConfiguration } from "./
      ↪ GlobalCollateralConfiguration.sol";
   import { GlobalCollateralConfiguration } from "./
      ↪ GlobalCollateralConfiguration.sol";
```

CVF-89 INFO

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

Recommendation The type for this argument should be more specific.

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

```
52 address collateral
```

CVF-90 INFO

- **Category** Procedural
- **Source** RiskMultipliersConfiguration.sol

Description We didn't review this file.

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

CVF-91 INFO

- **Category** Procedural
- **Source** LiquidationConfiguration.sol

Description We didn't review this file.

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

CVF-92 INFO

- **Category** Procedural
- **Source** AutoExchangeConfiguration.sol

Description We didn't review this file.

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

CVF-93 INFO

- **Category** Procedural
- **Source** BackstopLPConfiguration.sol

Description We didn't review this file.

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

CVF-94 FIXED

- **Category** Procedural
- **Source** BackstopLPConfiguration.sol

Recommendation This check should be performed earlier.

```
44 if (config.liquidationFee.gt(UNIT)) {
       revert Errors.InvalidBackstopConfiguration(config);
```

CVF-95 INFO

- **Category** Procedural
- **Source** InsuranceFundConfiguration.sol

Description We didn't review these files.

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

CVF-96 INFO

- **Category** Procedural
- **Source** CollateralPool.sol

Description We didn't review these files.

```
38 import { UD60x18, UNIT, ZERO } from "@prb/math/UD60x18.sol";
   import { SafeCastU256, SafeCastI256 } from "@voltz-protocol/util-
      ↪ contracts/src/helpers/SafeCast.sol";
40 import { SetUtil } from "@voltz-protocol/util-contracts/src/helpers/
      ↪ SetUtil.sol";
   import { FeatureFlag } from "@voltz-protocol/util-modules/src/
      ↪ storage/FeatureFlag.sol";
```

CVF-97 INFO

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

Recommendation The key type for these mappings should be more specific.

Client Comment *See CVF-89.*

```
72 mapping(address => uint256) collateralShares;
```

```
86 mapping(address quoteCollateral => SetUtil.AddressSet)
      ↪ supportingCollaterals;
```

CVF-98 INFO

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

Recommendation The type for the collateral arguments should be more specific.

Client Comment *See CVF-89.*

```
96     address quoteCollateral
```

```
274 function checkCap(Data storage self, address collateral) private
    ↪ view {
```

```
295 function getCollateralBalance(Data storage self, address collateral)
    ↪  internal view returns (uint256) {
```

```
304 function updateCollateralShares(Data storage self, address
    ↪ collateral, int256 sharesDelta) internal {
```

```
320 function _updateCollateralShares(Data storage self, address
    ↪ collateral, int256 sharesDelta) private {
```

```
393 function getParent(Data storage self, address collateral) internal
    ↪ view returns (address) {
```

```
397 function isQuoteCollateral(Data storage self, address collateral)
    ↪ internal view returns (bool) {
```

```
401 function isSupportingCollateral(Data storage self, address
    ↪ collateral) internal view returns (bool) {
```

```
407     address quoteCollateral,
        address supportingCollateral
```

```
420 function setCollateralParent(Data storage self, address collateral,
    ↪ address newParentCollateral) internal {
```

```
482     address collateralA,
        address collateralB
```

CVF-99 FIXED

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

Description The name is confusing, as it looks like a function that returns a boolean value.

Recommendation Consider renaming to “verifyActive” or “ensureActive”.

```
231 function checkActive(Data storage self) internal view {
```

CVF-100 FIXED

- **Category** Procedural
- **Source** CollateralPool.sol

Recommendation Brackets around “UNIT.sub(a)” are redundant.

```
390 return UNIT.sub((UNIT.sub(a)).mul(UNIT.sub(b)));
```

CVF-101 INFO

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

Recommendation The return type should be more specific.

Client Comment *See CVF-89.*

```
393 function getParent(Data storage self, address collateral) internal
    ↪ view returns (address) {
```

CVF-102 INFO

- **Category** Procedural
- **Source** AccountRBAC.sol

Description We didn’t review this file.

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

CVF-103 INFO

- **Category** Suboptimal
- **Source** AccountRBAC.sol

Description This check is redundant.

Recommendation Consider just returning “false” for non-existing permissions.

Client Comment *Check not removed due to concern regarding collisions.*

```
144 checkPermissionIsValid(permission);
```

CVF-104 INFO

- **Category** Suboptimal
- **Source** AccountRBAC.sol

Recommendation The “target != address(0)” check seems redundant.

Client Comment *Mindful this library is very sensitive.*

```
147 return target != address(0) && accountRBAC.permissions[target].
    ↪ contains(permission);
```

CVF-105 INFO

- **Category** Procedural
- **Source** Account.sol

Description We didn’t review these files.

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

```
23 import { SafeCastU256, SafeCastI256 } from "@voltz-protocol/util-
   ↪ contracts/src/helpers/SafeCast.sol";
import { UD60x18 } from "@prb/math/UD60x18.sol";
```

CVF-106 FIXED

- **Category** Procedural
- **Source** Account.sol

Description Here a file imports itself.

Recommendation Remove this import.

```
12 import { Account } from "./Account.sol";
```

CVF-107 INFO

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

Recommendation The key type for these mappings should be more specific.

Client Comment *See CVF-89.*

```
47 mapping(address quoteCollateral => SetUtil.UintSet markets)
       ↪ activeMarketsPerQuoteCollateral;
```

```
57 mapping(address => LiquidationBidQueues.Queues)
       ↪ liquidationBidQueuesPerBubble;
```

CVF-108 INFO

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

Description We didn't review these files.

```
19 import { SetUtil } from "@voltz-protocol/util-contracts/src/helpers/
       ↪ SetUtil.sol";
20 import { AccessError } from "@voltz-protocol/util-contracts/src/
       ↪ errors/AccessError.sol";
   import { UD60x18 } from "@prb/math/UD60x18.sol";
   import { SD59x18 } from "@prb/math/SD59x18.sol";
```

CVF-109 INFO

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

Recommendation The type for this field should be more specific.

Client Comment *See CVF-89.*

```
45 address quoteCollateral;
```

CVF-110 INFO

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

Recommendation The type for this field should be “IInstrument”.

Client Comment *For consistency, we will keep these types as addresses and check their interfaces when registered in storage.*

```
54 address instrumentAddress;
```

CVF-111 INFO

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

Recommendation The type for these arguments should be more specific.

Client Comment *See CVF-89, CVF-110.*

```
81 address instrumentAddress,
   address quoteCollateral,
```

CVF-112 INFO

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

Recommendation This code could be simplified by iterating “i” from 1 to “filledExposures.length-1” rather than from “0” to “filledExposures.length-2”.

```
151 for (uint256 i = 0; i + 1 < filledExposures.length; i++) {
        if (filledExposures[i].riskMatrixIndex >= filledExposures[i +
            ↪ 1].riskMatrixIndex) {
```

CVF-113 INFO

- **Category** Procedural
- **Source** GlobalCollateralConfiguration.sol

Description We didn’t review these files.

```
15 import { mulUDxUint, UD60x18 } from "@voltz-protocol/util-contracts/
       ↪ src/helpers/PrbMathHelper.sol";
   import { SignedMath } from "oz/utils/math/SignedMath.sol";
   import { SafeCastU256, SafeCastI256 } from "@voltz-protocol/util-
       ↪ contracts/src/helpers/SafeCast.sol";
```

```
21 import { IERC20 } from "@voltz-protocol/util-contracts/src/
       ↪ interfaces/IERC20.sol";
   import { ERC165Helper } from "@voltz-protocol/util-contracts/src/
       ↪ helpers/ERC165Helper.sol";
   import { Time } from "@voltz-protocol/util-contracts/src/helpers/
       ↪ Time.sol";
```

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

CVF-114 INFO

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

Recommendation The type for the collateral arguments should be more specific.

Client Comment *See CVF-89.*

```
44     address collateralAddress,
```

```
91 function exists(address collateral) internal view returns (Data
   ↪ storage globalCollateral) {
```

```
102 function load(address collateral) private pure returns (Data storage
   ↪  globalCollateral) {
```

```
177 function getCollateralDecimals(address collateralAddress) internal
   ↪ view returns (uint8) {
```

```
215 function getDecimalsDelta(address fromCollateral, address
   ↪ toCollateral) internal view returns (int8) {
```

CVF-115 FIXED

- **Category** Procedural
- **Source** GlobalCollateralConfiguration.sol

Recommendation These checks should be performed earlier, before loading the stored config.

```
56 if (!config.isStandardCollateral) {
       if (!ERC165Helper.safeSupportsInterface(config.collateralAdapter
          ↪ , type(ICollateralAdapter).interfaceId)) {
```

```
61     if (config.collateralAdapter != address(0)) {
```

```
66 if (config.withdrawalTvlPercentageLimit.gt(UNIT)) {
```

CVF-116 INFO

- **Category** Procedural
- **Source** GlobalCollateralConfiguration.sol

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

Recommendation Consider treating all token amounts as integers.

```
72 uint8 collateralDecimals = IERC20(collateralAddress).decimals();
```

CVF-117 INFO

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

Description Overflow is possible when converting to “int8”.

Recommendation Consider using a wider type.

Client Comment *Added documentation to specify the function requirements. This shouldn’t be the case with any registered tokens.*

```
219 return int8(decimalsTo) - int8(decimalsFrom);
```

CVF-118 INFO

- **Category** Procedural
- **Source** RiskMatrix.sol

Description We didn’t review these files.

```
16 import { SD59x18, ZERO as ZERO_sd, UNIT as UNIT_sd, abs } from "@prb
      ↪ /math/SD59x18.sol";
   import { SetUtil } from "@voltz-protocol/util-contracts/src/helpers/
      ↪ SetUtil.sol";
```

CVF-119 INFO

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

Recommendation The key type for this mapping should be more specific.

Client Comment *See CVF-89.*

```
34 mapping(address quoteCollateral => SetUtil.UintSet riskBlocks)
      ↪ blocksOfQuoteCollateral;
```

CVF-120 FIXED

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

Recommendation The storage slot address should be a named constant.

Client Comment *Changed to constant keccak256(abi.bytes("xyz.reya.Signature"));*

```
27 bytes32 s = keccak256(abi.encode("xyz.reya.Signature"));
```

CVF-121 FIXED

- **Category** Suboptimal
- **Source** Signature.sol

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

Recommendation Consider reusing the written value: return ++signatureObject.sigNonces[accountOwner];

```
35 signatureObject.sigNonces[accountOwner] += 1;
   return signatureObject.sigNonces[accountOwner];
```

CVF-122 FIXED

- **Category** Suboptimal
- **Source** IdStore.sol

Description Using ‘encode‘ is suboptimal.

Recommendation Consider using ‘encodePacked()‘.

80
```
bytes32 key = keccak256(abi.encode(RISK_BLOCK_ID, collateralPoolId))
   ↪ ;
```

88
```
bytes32 key = keccak256(abi.encode(RISK_BLOCK_ID, collateralPoolId))
   ↪ ;
```

CVF-123 INFO

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

Description This event is emitted even if nothing actually changed.

Client Comment *Checking if something actually changed introduces complexity.*

46
```
emit Events.InstrumentRegistrationUpdated({ instrumentAddress:
   ↪ instrumentAddress, isRegistered: isRegistered });
```

CVF-124 FIXED

- **Category** Suboptimal
- **Source** Exchange.sol

Recommendation This could be simplified as: return ownerExchangePassBalance > 0;

85
```
if (ownerExchangePassBalance > 0) {
    return true;
}
```

89
```
return false;
```

CVF-125 INFO

- **Category** Procedural
- **Source** BackstopLiquidationModule.sol

Description We didn't review this file.

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

CVF-126 INFO

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

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

Client Comment *See CVF-89.*

```
32 address quoteCollateral,
```

CVF-128 INFO

- **Category** Suboptimal
- **Source** CommonLiquidationModule.sol

Recommendation Consider including the actual response as an error parameter, so the hook will be able to return the reason why liquidation shouldn't be performed.

Client Comment *This error is only given under specific conditions, so the cause is obvious.*

```
69 revert Errors.InvalidPreLiquidationHookResponse();
```

CVF-129 INFO

- **Category** Suboptimal
- **Source** CommonLiquidationModule.sol

Recommendation Consider including the actual response as an error parameter, so the hook will be able to return the reason why liquidation should be reverted.

Client Comment *This error is only given under specific conditions, so the cause is obvious.*

```
130 revert Errors.InvalidPostLiquidationHookResponse();
```

CVF-130 INFO

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

Recommendation The type for this argument should be “IERC20”.

Client Comment *See CVF-89.*

```
29 address quoteCollateral,
```

CVF-131 INFO

- **Category** Suboptimal
- **Source** DutchLiquidationModule.sol

Recommendation It would be more efficient to pass a single array of structs with two fields, rather than two parallel arrays.

Client Comment *Minimal effect.*

```
30 uint128[] memory marketIds,
   bytes[] memory inputs
```

CVF-133 INFO

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

Recommendation The type for this argument should be “IERC20”.

Client Comment *See CVF-89.*

```
32 address quoteCollateral,
```

CVF-135 INFO

- **Category** Procedural
- **Source** AaveCollateralAdapter.sol

Description We didn’t review this function.

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

CVF-136 INFO

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

Recommendation The type for this variable should be “IACollateral”.

Client Comment *See CVF-110.*

```
16 address internal _asset;
```

CVF-137 INFO

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

Recommendation The argument type should be “IACollateral”.

Client Comment *See CVF-110.*

```
18 constructor(address assetCollateralAddress) {
```

CVF-138 INFO

- **Category** Procedural
- **Source** LidoCollateralAdapter.sol

Description We didn't review this function.

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

CVF-139 INFO

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

Recommendation The type for this variable should be "IStEth".

Client Comment *See CVF-110.*

```
16 address internal _asset;
```

CVF-140 INFO

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

Recommendation The argument type should be "IStEth".

Client Comment *See CVF-110.*

```
18 constructor(address assetCollateralAddress) {
```

CVF-141 INFO

- **Category** Procedural
- **Source** AccountModule.sol

Description We didn't review these files.

```
25 import { SetUtil } from "@voltz-protocol/util-contracts/src/helpers/
      ↪ SetUtil.sol";
   import { SD59x18 } from "@prb/math/SD59x18.sol";
   import { UD60x18 } from "@prb/math/UD60x18.sol";
```

CVF-142 INFO

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

Description This function should emit some event.

Client Comment *The events emitted in AccountRBAC are sufficient.*

```
54 function setCustomImMultiplier(uint128 accountId, UD60x18
   ↪ imMultiplier) external override {
```

```
103 function grantAccountPermission(uint128 accountId, bytes32
   ↪ permission, address user) external override {
```

```
113 function revokeAccountPermission(uint128 accountId, bytes32
   ↪ permission, address user) external override {
```

```
123 function renounceAccountPermission(uint128 accountId, bytes32
   ↪ permission) external override {
```

CVF-143 INFO

- **Category** Documentation
- **Source** AccountModule.sol

Description In case there is no pool for the account, this function returns zero.

Recommendation Consider clearly documenting this behavior or reverting in such a case.

Client Comment *Behaviour is documented in the interface.*

```
208 function getCollateralPoolIdOfAccount(uint128 accountId) external
   ↪ view override returns (uint128) {
```

CVF-144 INFO

- **Category** Procedural
- **Source** AccountTokenModule.sol

Description We didn't review these files.

```
12 import { SafeCastU256 } from "@voltz-protocol/util-contracts/src/
      ↪ helpers/SafeCast.sol";
   import { NFT } from "@voltz-protocol/util-modules/src/modules/
      ↪ NftModule.sol";
   import { OwnableStorage } from "@voltz-protocol/util-contracts/src/
      ↪ storage/OwnableStorage.sol";
```

CVF-145 INFO

- **Category** Suboptimal
- **Source** AccountTokenModule.sol

Description Using the owner as the address to be notifying is inflexible.

Recommendation Consider introducing a separate variable of type "IAccountModule", probably immutable, for the address to be notified.

Client Comment *Maintaining another role adds complexity.*

```
27 IAccountModule(OwnableStorage.getOwner()).notifyAccountTransfer(from
      ↪ , to, tokenId.to128());
```

CVF-146 INFO

- **Category** Procedural
- **Source** AssociatedSystemsModule.sol

Description We didn't review this file.

```
10 import { BaseAssociatedSystemsModule } from "@voltz-protocol/util-
      ↪ modules/src/modules/BaseAssociatedSystemsModule.sol";
```

CVF-147 INFO

- **Category** Procedural
- **Source** AssociatedSystemsModule.sol

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

Client Comment *See CVF-89*.

```
16 contract AssociatedSystemsModule is BaseAssociatedSystemsModule { }
```

CVF-148 INFO

- **Category** Procedural
- **Source** CollateralModule.sol

Description We didn't review this file.

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

CVF-149 INFO

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

Recommendation The type for the collateral arguments should be "IERC20".

Client Comment *See CVF-89*.

```
35     address collateralAddress,
```

```
49 function getGlobalCollateralConfig(address collateralAddress)
```

```
63     address collateralAddress,
```

```
85     address collateralAddress
```

CVF-150 INFO

- **Category** Procedural
- **Source** CollateralPoolModule.sol

Description We didn't review this file.

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

CVF-151 INFO

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

Recommendation The type for the collateral argument should be "IERC20".

Client Comment *See CVF-89.*

```
73 function getCollateralPoolBalance(uint128 collateralPoolId, address
   ↪ collateral) external view returns (uint256) {
```

CVF-153 INFO

- **Category** Readability
- **Source** ExecutionModule.sol

Recommendation Should be "else if".

Client Comment *We prefer avoiding the "if else" nesting pattern.*

```
161 if (command.commandType == CommandType.MatchOrder) {
```

CVF-154 INFO

- **Category** Readability
- **Source** ExecutionModule.sol

Recommendation Should be "else revert".

Client Comment *We prefer avoiding the "if else" nesting pattern.*

```
232 revert Errors.InvalidCommandType(command.commandType);
```

CVF-155 INFO

- **Category** Procedural
- **Source** InstrumentModule.sol

Description We didn't review these files.

```
21 import { ERC165Helper } from "@voltz-protocol/util-contracts/src/
   ↪ helpers/ERC165Helper.sol";
```

```
23 import { SetUtil } from "@voltz-protocol/util-contracts/src/helpers/
   ↪ SetUtil.sol";
   import { SafeCastU256, SafeCastI256 } from "@voltz-protocol/util-
   ↪ contracts/src/helpers/SafeCast.sol";
```

CVF-156 INFO

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

Recommendation The type for the "quoteCollateral" should be "IERC20".

Client Comment *See CVF-89.*

```
50 function registerMarket(address quoteCollateral, string memory name)
   ↪  external override returns (uint128 marketId) {
```

CVF-157 INFO

- **Category** Suboptimal
- **Source** InstrumentModule.sol

Recommendation This check should be performed earlier.

Client Comment *Moving this check has minor impact but affects consistency.*

```
57 if (quoteCollateral == address(0)) {
       revert Errors.ZeroQuoteCollateralAddress();
```

CVF-158 INFO

- **Category** Procedural
- **Source** InstrumentRegistrarModule.sol

Description We didn't review this file.

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

CVF-159 INFO

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

Recommendation The type for the "instrumentAddress" arguments should be "IInstrument".

Client Comment *See CVF-110.*

```
20 function setInstrumentRegistrationFlag(address instrumentAddress,
   ↪ bool isRegistered) external {
```

```
31 function isInstrumentRegistered(address instrumentAddress) external
   ↪ view returns (bool isRegisteredFlag) {
```

CVF-160 INFO

- **Category** Procedural
- **Source** ProtocolConfigurationModule.sol

Description We didn't review this file.

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

CVF-161 INFO

- **Category** Procedural
- **Source** RiskConfigurationModule.sol

Description We didn't review these files.

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

CVF-162 FIXED

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

Recommendation The type for the collateral argument should be "IERC20".

Client Comment *See CVF-89.*

```
38 function isCollateralInBubbleExhausted(uint128 accountId, address
   ↪ inCollateral) external view returns (bool) {
```

```
47     address outCollateral,
       address quoteCollateral
```

```
66     address inCollateral
```

CVF-164 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-166 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-168 INFO

- **Category** Procedural
- **Source** CreateAccount.sol

Description We didn't review this file.

```
13 import { AssociatedSystem } from "@voltz-protocol/util-modules/src/
   ↪ storage/AssociatedSystem.sol";
```

CVF-169 INFO

- **Category** Procedural
- **Source** EditCollateral.sol

Description We didn't review these files.

```
19 import { ERC20Helper, IERC20 } from "@voltz-protocol/util-contracts/
   ↪ src/token/ERC20Helper.sol";
20 import { SafeCastU256 } from "@voltz-protocol/util-contracts/src/
   ↪ helpers/SafeCast.sol";
   import { Timer } from "@voltz-protocol/util-contracts/src/helpers/
   ↪ Timer.sol";
```

CVF-170 FIXED

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

Description The word “edit” sounds odd in this context.

Recommendation Consider renaming to something like “TransferCollateral”.

```
26 library EditCollateral {
```

CVF-171 INFO

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

Recommendation The type for the “collateral” arguments should be “IERC20”.

Client Comment *See CVF-89*.

```
45 function deposit(Account.Data storage account, address collateral,
   ↪ uint256 collateralAmount) internal {
```

```
81 function withdraw(Account.Data storage account, address collateral,
   ↪ uint256 collateralAmount) internal {
```

CVF-172 FIXED

- **Category** Documentation
- **Source** EditCollateral.sol

Recommendation Typo: “if account if”.

```
87 // Check if account if backstop lp.
```

CVF-173 FIXED

- **Category** Suboptimal
- **Source** EditCollateral.sol

Description The expression “collateralPool.bbackstopLPConfig()” is calculated twice.

Recommendation Consider calculating once and reusing.

117
```
uint128 backstopLpAccountId = collateralPool.backstopLPConfig().
    ↪ accountId;
```

134
```
    block.timestamp + collateralPool.backstopLPConfig().
        ↪ withdrawCooldownDurationInSeconds,
```

CVF-174 INFO

- **Category** Procedural
- **Source** AccountActiveMarket.sol

Description We didn’t review this file.

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

CVF-175 INFO

- **Category** Procedural
- **Source** AccountExposure.sol

Description We didn't review these files.

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

```
30 import { IERC20 } from "@voltz-protocol/util-contracts/src/
      ↪ interfaces/IERC20.sol";
   import { SafeCastU256, SafeCastI256 } from "@voltz-protocol/util-
      ↪ contracts/src/helpers/SafeCast.sol";
   import { SetUtil } from "@voltz-protocol/util-contracts/src/helpers/
      ↪ SetUtil.sol";
   import { mulUDxUint, maxUDxUD } from "@voltz-protocol/util-contracts
      ↪ /src/helpers/PrbMathHelper.sol";
   import { convert as convert_sd, SD59x18, ZERO as ZERO_sd, sd } from
      ↪ "@prb/math/SD59x18.sol";
   import { UD60x18, ZERO, convert as convert_ud, unwrap, UD60x18, ud }
      ↪  from "@prb/math/UD60x18.sol";
   import { SignedMath } from "oz/utils/math/SignedMath.sol";
```

CVF-176 INFO

- **Category** Suboptimal
- **Source** AccountExposure.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.

```
235 uint8 quoteDecimals = GlobalCollateralConfiguration.
       ↪ getCollateralDecimals(collateral);
```

CVF-177 FIXED

- **Category** Procedural
- **Source** AccountAutoExchange.sol

Recommendation These imports should be merged.

```
10 import { MarginInfo, CollateralInfo, AutoExchangeAmounts } from "../
      ↪ DataTypes.sol";
```

```
19 import { ExchangeInfo } from "../DataTypes.sol";
```

CVF-178 FIXED

- **Category** Procedural
- **Source** AccountAutoExchange.sol

Recommendation Duplicated import.

```
12 import { AccountExposure } from "./AccountExposure.sol";
```

```
18 import { AccountExposure } from "./AccountExposure.sol";
```

CVF-179 INFO

- **Category** Procedural
- **Source** AccountAutoExchange.sol

Description We didn't review these files.

```
21 import { SetUtil } from "@voltz-protocol/util-contracts/src/helpers/
      ↪ SetUtil.sol";
   import { mulUDxUint, divUintUD } from "@voltz-protocol/util-
      ↪ contracts/src/helpers/PrbMathHelper.sol";
   import { UD60x18, UNIT } from "@prb/math/UD60x18.sol";
   import { SafeCastU256, SafeCastI256 } from "@voltz-protocol/util-
      ↪ contracts/src/helpers/SafeCast.sol";
```

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

CVF-180 INFO

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

Recommendation The type for the collateral arguments should be more specific.

Client Comment *See CVF-89.*

```
49      address quoteCollateral
```

```
97  function getRawMaxQuoteToCover(Account.Data storage account, address
    ↪  inCollateral) private view returns (uint256) {
```

```
147     address inCollateral
```

```
172     address outCollateral,
```

```
209     address outCollateral,
210     address quoteCollateral,
```

CVF-181 INFO

- **Category** Procedural
- **Source** AccountAutoExchange.sol

Recommendation Brackets are redundant.

Client Comment *Kept for readability.*

```
100 bool isMismatched = (usdNodeMarginInfo.maintenanceDelta < 0) && (
    ↪ quoteTokenMarginInfo.liquidationDelta < 0);
```

CVF-182 INFO

- **Category** Suboptimal
- **Source** AccountChecks.sol

Description This variable is redundant, as its value is used only once.

Recommendation Consider removing this variable and using the expression instead.

Client Comment *Kept for readability.*

```
21 bool isAboveAdl = marginInfo.adlDelta >= 0;
```

```
31 bool isAboveLm = marginInfo.liquidationDelta >= 0;
```

```
41 bool isAboveIm = marginInfo.initialDelta >= 0;
```

```
51 bool isBelowAdl = marginInfo.adlDelta < 0;
```

```
61 bool isBelowLm = marginInfo.liquidationDelta < 0;
```

```
71 bool isBelowDutch = marginInfo.dutchDelta < 0;
```

```
81 bool isBelowMmr = marginInfo.maintenanceDelta < 0;
```

```
126 bool isSolvent = marginInfo.marginBalance > 0;
```

```
136 bool isRealBalancePositive = collateralInfo.realBalance >= 0;
```

CVF-183 INFO

- **Category** Procedural
- **Source** AccountBackstopADL.sol

Description We didn't review these files.

```
21 import { SafeCastU256 } from "@voltz-protocol/util-contracts/src/
   ↪ helpers/SafeCast.sol";
import { mulUDxUint, mulUDxInt, divUintUint } from "@voltz-protocol/
   ↪ util-contracts/src/helpers/PrbMathHelper.sol";
import { UD60x18, UNIT, ZERO } from "@prb/math/UD60x18.sol";
import { SignedMath } from "oz/utils/math/SignedMath.sol";
import { SetUtil } from "@voltz-protocol/util-contracts/src/helpers/
   ↪ SetUtil.sol";
```

CVF-184 INFO

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

Recommendation The type for the collateral arguments should be more specific.

Client Comment *See CVF-89*.

```
48  address quoteCollateral,
```

```
126 address quoteCollateral,
```

```
161 address quoteCollateral,
```

```
189 address quoteCollateral,
```

```
249 address quoteCollateral,
```

CVF-185 INFO

- **Category** Suboptimal
- **Source** AccountBackstopADL.sol

Description The expression “keeperReward.toInt()” is calculated seceral times.

Recommendation Consider calculating once and reusing.

```
197 AccountCollateral.updateBalance(liquidatedAccount.id,
       ↪ quoteCollateral, -keeperReward.toInt());
    AccountCollateral.updateBalance(keeper.id, quoteCollateral,
       ↪ keeperReward.toInt());
```

```
200 availableFunds -= keeperReward.toInt();
```

CVF-186 INFO

- **Category** Suboptimal
- **Source** AccountBackstopADL.sol

Recommendation It would be more efficient to return a single array of structs with two fields, rather than two parallel arrays.

Client Comment *Kept for readability.*

```
214 returns (bool[] memory isMarketActive, int256[] memory supportedUPnL
    ↪ )
```

CVF-187 INFO

- **Category** Suboptimal
- **Source** AccountBackstopADL.sol

Recommendation This array is redundant. Just use “supportedUPnL” array instead.

Client Comment *Kept for readability.*

```
219 uint256[] memory absUPnLs = new uint256[](markets.length);
```

CVF-188 INFO

- **Category** Suboptimal
- **Source** AccountBackstopADL.sol

Description The expression “-existingFunds” is calculated on every loop iteration.

Recommendation Consider calculating once before the loop.

Client Comment *Kept for readability.*

```
236 supportedUPnL[i] = mulUDxInt(divUintUint(absUPnLs[i], absUPnLSum), -
    ↪ existingFunds);
```

CVF-189 INFO

- **Category** Procedural
- **Source** AccountLiquidation.sol

Description We didn't review these files.

```
37 import { UD60x18, mulUDxUint } from "@voltz-protocol/util-contracts/
      ↪ src/helpers/PrbMathHelper.sol";
   import { SafeCastU256, SafeCastI256 } from "@voltz-protocol/util-
      ↪ contracts/src/helpers/SafeCast.sol";
   import { SetUtil } from "@voltz-protocol/util-contracts/src/helpers/
      ↪ SetUtil.sol";
40 import { UD60x18, UNIT, ZERO, ud, convert } from "@prb/math/UD60x18.
      ↪ sol";
   import { SD59x18 } from "@prb/math/SD59x18.sol";
```

```
43 import { ERC165Helper } from "@voltz-protocol/util-contracts/src/
      ↪ helpers/ERC165Helper.sol";
```

CVF-190 INFO

- **Category** Suboptimal
- **Source** AccountLiquidation.sol

Recommendation It would be more efficient to pass a single array of structs with two fields, rather than two parallel arrays. This would also make the length check unnecessary.

Client Comment *Kept for readability.*

```
80 uint128[] memory marketIds,
   bytes[] memory inputs,
```

CVF-191 INFO

- **Category** Suboptimal
- **Source** AccountLiquidation.sol

Description The expression “marketNormalizedExposure.abs().intoUD60×17” is calculated twice.

Recommendation Consider calculating once and reusing.

```
197 numerator = numerator.add(marketNormalisedExposure.abs().intoUD60x18
        ↪ ().mul(marketPSlippage));
    denominator = denominator.add(marketNormalisedExposure.abs().
        ↪ intoUD60x18());
```

CVF-192 FIXED

- **Category** Procedural
- **Source** AccountLiquidation.sol

Description This comment duplicates the formula in the comment above the function.

Recommendation Consider removing it.

Client Comment *Removed.*

```
206 // score = 1/n * [w * (1 - d) + (1 - w) * pSlippage];
```

CVF-193 INFO

- **Category** Procedural
- **Source** AccountLiquidation.sol

Recommendation This check should be done before enqueueing a new bid.

Client Comment *Minor impact on gas but adds complexity.*

```
295 if (liquidationBidQueue.size() > liquidationConfig.maxBidsInQueue) {
        revert Errors.LiquidationBidQueueOverflow();
```

CVF-194 INFO

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

Recommendation The type for the collateral arguments should be more specific.

Client Comment *See CVF-89.*

329
```
address collateral,
```

425
```
address quoteCollateral,
```

493
```
address quoteCollateral,
```

CVF-195 FIXED

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

Description This should be executed only when there is a keeper, i.e. inside the "if" statement above.

Client Comment *Moved inside if (bidSubmissionKeeperId != 0)*

365
```
rewards.liquidator -= rewards.keeper;
```

CVF-196 INFO

- **Category** Suboptimal
- **Source** AccountLiquidation.sol

Description This check is redundant.

Recommendation Consider removing it. It would be neede in case negative health values would be possible, but this is technically impossible, as "health" is unsigned.

Client Comment *Agreed but this constraint is not obvious.*

406
```
// note, this should never be the case, so check might be redandunt
if (dDutch.gt(liquidationConfig.dMax)) {
```

CVF-198 INFO

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

Recommendation Events are usually named via nouns, such as “AccountOwner” or “GrantedAccountPermission”.

```
37  event AccountOwnerUpdated(uint128 indexed accountId, address indexed
        ↪  newOwner, uint256 blockTimestamp);

46  event AccountPermissionGranted(

57  event AccountPermissionRevoked(

68  event AccountCollateralSharesUpdated(

101 event CollateralBaseConfigurationUpdated(

110 event CollateralParentConfigurationUpdated(

119 event CollateralPoolOwnerUpdated(uint128 indexed collateralPoolId,
        ↪ address newOwner, uint256 blockTimestamp);

121 event CollateralPoolCollateralSharesUpdated(

129 event RiskMultipliersUpdated(uint128 indexed collateralPoolId,
        ↪ RiskMultipliers newConfig, uint256 blockTimestamp);

131 event LiquidationConfigurationUpdated(

135 event InsuranceFundConfigurationUpdated(

139 event BackstopLPConfigurationUpdated(

143 event AutoExchangeConfigurationUpdated(

147 event LimitConfigurationUpdated(uint128 indexed collateralPoolId,
        ↪ LimitConfig newConfig, uint256 blockTimestamp);

151 event GlobalCollateralConfigurationUpdated(
```

(158, 160, 167, 174, 185, 193, 202, 210, 219, 234, 253)

CVF-199 INFO

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

Recommendation The type for the collateral parameters should be more specific.

Client Comment *See CVF-89.*

```
69  uint128 indexed accountId, address indexed collateral, int256
       ↪ sharesDelta, uint256 blockTimestamp
```

```
102 uint128 indexed collateralPoolId, address collateralAddress,
       ↪ CollateralConfig newConfig, uint256 blockTimestamp
```

```
112 address collateralAddress,
```

```
122 uint128 indexed collateralPoolId, address indexed collateral, int256
       ↪  sharesDelta, uint256 blockTimestamp
```

```
152 address collateralAddress, uint8 collateralDecimals,
       ↪ GlobalCollateralConfig newConfig, uint256 blockTimestamp
```

```
205 address quoteCollateral,
```

```
236 address indexed collateral,
```

```
255 address indexed collateral,
```

CVF-200 INFO

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

Recommendation The type for the instrument parameters should be more specific.

Client Comment *See CVF-110.*

```
160 event InstrumentRegistrationUpdated(address instrumentAddress, bool
       ↪ isRegistered);
```

```
193 event InstrumentRegistrationFlagSet(address instrumentAddress, bool
       ↪ isRegistered, uint256 blockTimestamp);
```

CVF-201 INFO

- **Category** Procedural
- **Source** PriceHelpers.sol

Description We didn't review these files.

```
10 import { UD60x18, UNIT } from "@prb/math/UD60x18.sol";
   import { mulUDxInt } from "@voltz-protocol/util-contracts/src/
      ↪ helpers/PrbMathHelper.sol";
   import { DecimalMath } from "@voltz-protocol/util-contracts/src/
      ↪ helpers/DecimalMath.sol";
   import { SafeCastU256, SafeCastI256 } from "@voltz-protocol/util-
      ↪ contracts/src/helpers/SafeCast.sol";
```

CVF-202 INFO

- **Category** Suboptimal
- **Source** PriceHelpers.sol

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

Recommendation Consider treating all token amounts as integers.

Client Comment *The logic needs decimals for exchange purposes.*

```
67 } else if (decimalsDelta > 0) {
       return mulUDxInt(price, DecimalMath.changeDecimals(
          ↪ amountOfCollateralA, decimalsDelta, roundUp));
```

```
70     return DecimalMath.changeDecimals(mulUDxInt(price,
          ↪ amountOfCollateralA), decimalsDelta, roundUp);
```

CVF-203 INFO

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

Description We didn't review this file.

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

CVF-204 FIXED

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

Description Unlike other constants in the same enum, this constant isn't documented.

Recommendation Consider documenting for consistency.

17 `PropagateCashflow`

CVF-205 FIXED

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

Description Unlike other fields in the same struct, this field isn't documented.

Recommendation Consider documenting for consistency.

72 `address collateral;`

95 `int256 netDeposits;`

CVF-206 INFO

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

Recommendation The type for this field should be more specific.

Client Comment *See CVF-89.*

72 `address collateral;`

107 `address quoteCollateral;`

357
```
address collateral;
address inCollateral;
```

421 `address collateralAddress;`

452 `address collateralAddress;`

472 `address collateralAdapter;`

487 `address collateralAddress;`

CVF-207 INFO

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

Recommendation The type for this fields should be “ILiquidationHook”.

Client Comment *See CVF-110.*

```
105 address hookAddress;
```

CVF-208 FIXED

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

Description The number format for these fields is unclear.

Recommendation Consider documenting.

```
317 uint256 liquidationMarginRequirement;
    uint256 initialMarginRequirement;
    uint256 maintenanceMarginRequirement;
320 uint256 dutchMarginRequirement;
    uint256 adlMarginRequirement;
    uint256 initialBufferMarginRequirement;
```

CVF-209 FIXED

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

Description This comment seems irrelevant.

Recommendation Consider removing it.

```
490 * @notice If the address is ZERO_ADDRESS, it represents USD.
```

CVF-210 INFO

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

Description We didn't review these files.

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

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

CVF-211 INFO

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

Description This library consists only of errors.

Recommendation Consider moving the errors into the top level and removing this library.

Client Comment *We prefer keeping these separate from the logic.*

```
31 library Errors {
```

CVF-212 INFO

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

Recommendation The type for the collateral parameters should be more specific.

Client Comment *See CVF-89.*

```
219 error QuoteCollateralCannotBecomeSupportingCollateral(uint128
    ↪ collateralPoolId, address collateral);

221 error InvalidNewParentCollateral(uint128 collateralPoolId, address
    ↪ newParentCollateral);

227 error CollateralIsNotQuote(uint128 collateralPoolId, address
    ↪ collateral);

231 error ZeroDeposit(uint128 accountId, address collateral);

233 error ZeroWithdraw(uint128 accountId, address collateral);

308 error AccountNotEligibleForAutoExchange(uint128 accountId, address
    ↪ inCollateral);

310 error WithinBubbleCoverageNotExhausted(uint128 accountId, address
    ↪ inCollateral, address collateral);

312 error SameQuoteAndcollateral(uint128 accountId, address inCollateral
    ↪ );

378 error CollateralDepositDisabled(uint128 collateralPoolId, address
    ↪ collateral);

387 error UnlinkedCollaterals(uint128 collateralPoolId, address
    ↪ collateral, address baseCollateral);

394 error CollateralNotConfigured(uint128 collateralPoolId, address
    ↪ collateral);

411 error collateralWithdrawLimitReached(address collateral, uint32
    ↪ windowStartTimestamp);

417     uint128 collateralPoolId, address collateralAddress, address
        ↪ invalidParentCollateralAddress
```

(424, 434, 450, 460, 470)

CVF-213 INFO

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

Recommendation The parameter type should be more specific.

Client Comment *See CVF-110.*

```
251 error IncorrectMarketInterface(address market);
```

CVF-214 INFO

- **Category** Suboptimal
- **Source** Errors.sol

Recommendation Logging timestamps is redundant as they can be retrieved from log events for free.

Client Comment *See CVF-47.*

```
330 error BackstopLpWithdrawPeriodInactive(uint128 backstopLpAccountId,
    ↪ uint256 blockTimestamp);
```

```
339 error AccountIsNotBackstopLp(uint128 accountId, uint128
    ↪ backstopLpAccountId, uint256 blockTimestamp);
```

```
348     uint256 backstopLpAccountId, uint256
        ↪ withdrawPeriodStartTimestamp, uint256 blockTimestamp
```

```
356 error BackstopLpWithdrawPeriodAlreadyActive(uint256
    ↪ backstopLpAccountId, uint256 blockTimestamp);
```

CVF-215 INFO

- **Category** Suboptimal
- **Source** Errors.sol

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

```
366 error InvalidPreLiquidationHookResponse();
```

```
371 error InvalidPostLiquidationHookResponse();
```

```
483 error SignatureInvalid();
```

```
485 error SignatureExpired();
```

CVF-216 INFO

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

Recommendation The parameter type should be more specific.

Client Comment *See CVF-110.*

```
481 error InstrumentNotFound(address instrumentAddress);
```

CVF-217 INFO

- **Category** Procedural
- **Source** IBackstopLiquidationModule.sol

Description We didn't review this file.

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

CVF-218 INFO

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

Recommendation The type for this argument should be more specific.

Client Comment *See CVF-89*.

```
28 address quoteCollateral,
```

CVF-219 INFO

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

Recommendation It would be more efficient to pass a single array of structs with two fields, rather than two parallel arrays. This would also make the length check unnecessary.

Client Comment *Keeping for readability*.

```
29 uint128[] memory marketIds,
30 bytes[] memory inputs
```

CVF-220 INFO

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

Recommendation The type for this argument should be more specific.

Client Comment *See CVF-89*.

```
31 address quoteCollateral,
```

CVF-221 INFO

- **Category** Procedural
- **Source** IACollateral.sol

Description We didn't review this file.

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

CVF-222 INFO

- **Category** Procedural
- **Source** IInstrument.sol

Description We didn't review these files.

```
17 import { IERC165 } from "@voltz-protocol/util-contracts/src/
      ↪ interfaces/IERC165.sol";
   import { UD60x18 } from "@prb/math/UD60x18.sol";
   import { SD59x18 } from "@prb/math/SD59x18.sol";
```

CVF-223 FIXED

- **Category** Documentation
- **Source** IInstrument.sol

Description This argument isn't documented.

Recommendation Consider documenting.

```
89 LiquidationType liquidationType,
```

CVF-224 INFO

- **Category** Procedural
- **Source** ILiquidationHook.sol

Description We didn't review this file.

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

CVF-225 INFO

- **Category** Procedural
- **Source** IStEth.sol

Description We didn't review this file.

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

CVF-226 INFO

- **Category** Procedural
- **Source** IAccountTokenModule.sol

Description We didn't review this file.

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

CVF-227 INFO

- **Category** Procedural
- **Source** IAccountTokenModule.sol

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

```
15 interface IAccountTokenModule is INftModule { }
```

CVF-228 INFO

- **Category** Suboptimal
- **Source** IAutoExchangeConfigurationModule.sol

Recommendation This function should emit some event and this event should be declared in this interface.

Client Comment *These module functions are wrappers for library functions that emit relevant events.*

```
26 function configureAutoExchange(uint128 collateralPoolId,
     ↪ AutoExchangeConfig memory config) external;
```

CVF-229 INFO

- **Category** Procedural
- **Source** IAccountModule.sol

Description We didn't review this file.

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

CVF-230 INFO

- **Category** Procedural
- **Source** IAccountModule.sol

Recommendation This event should be declared in this interface.

Client Comment *The event is declared in Events library,*

```
21 * Emits a {AccountCreated} event.
```

CVF-231 INFO

- **Category** Suboptimal
- **Source** IAccountModule.sol

Recommendation These functions should emit some events and these events should be declared in this interface.

Client Comment *These module functions are wrappers for library functions that emit relevant events.*

```
28 function setCustomImMultiplier(uint128 accountId, UD60x18
   ↪ imMultiplier) external;
```

```
66 function grantAccountPermission(uint128 accountId, bytes32
   ↪ permission, address user) external;
```

```
80 function revokeAccountPermission(uint128 accountId, bytes32
   ↪ permission, address user) external;
```

```
89 function renounceAccountPermission(uint128 accountId, bytes32
   ↪ permission) external;
```

```
138 function activateFirstMarketForAccount(uint128 accountId, uint128
   ↪ marketId) external;
```

```
212 function announceBackstopLpWithdraw(uint128 accountId) external;
```

CVF-232 FIXED

- **Category** Documentation
- **Source** IAccountModule.sol

Description The meaning of the word “first” in the name is unclear.

Recommendation Consider explaining in the documentation comment.

```
138 function activateFirstMarketForAccount(uint128 accountId, uint128
   ↪ marketId) external;
```

CVF-233 FIXED

- **Category** Documentation
- **Source** IAccountModule.sol

Description The structure and the semantics of the returned value are unclear.

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

```
207 returns (int256[] memory);
```

CVF-234 INFO

- **Category** Suboptimal
- **Source** IAutoExchangeModule.sol

Recommendation This function should emit some event and this event should be declared in this interface.

Client Comment *These module functions are wrappers for library functions that emit relevant events.*

```
22 function triggerAutoExchange(TriggerAutoExchangeInput memory input)
      ↪ external returns (AutoExchangeAmounts memory);
```

CVF-235 INFO

- **Category** Procedural
- **Source** ICollateralAdapter.sol

Description We didn't review this file.

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

CVF-236 INFO

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

Recommendation The returned type should be more specific.

Client Comment *See CVF-89.*

```
13 function asset() external view returns (address
   ↪ assetCollateralAddress);
```

CVF-237 INFO

- **Category** Suboptimal
- **Source** ICollateralModule.sol

Recommendation These functions should emit some events and these events should be declared in this interface.

Client Comment *These module functions are wrappers for library functions that emit relevant events.*

```
31 function setGlobalCollateralConfig(address collateralAddress,
   ↪ GlobalCollateralConfig memory config) external;
```

```
51 function setCollateralConfig(
```

CVF-238 INFO

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

Recommendation The type for these arguments should be more specific.

Client Comment *See CVF-89.*

```
53 address collateralAddress,
   CollateralConfig memory baseConfig,
```

```
66 address collateralAddress
```

CVF-239 INFO

- **Category** Suboptimal
- **Source** IInsuranceFundConfigurationModule.sol

Recommendation This function should emit some event and this event should be declared in this interface.

Client Comment *These module functions are wrappers for library functions that emit relevant events.*

```
26 function configureCollateralPoolInsuranceFund(
```

CVF-240 INFO

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

Recommendation The type for the "instrumentAddress" arguments should be "IInstrument".

Client Comment *See CVF-110.*

```
27 function setInstrumentRegistrationFlag(address instrumentAddress,
   ↪ bool isRegistered) external;
```

```
35 function isInstrumentRegistered(address instrumentAddress) external
   ↪ view returns (bool isRegisteredFlag);
```

CVF-241 INFO

- **Category** Suboptimal
- **Source** IInstrumentRegistrarModule.sol

Recommendation This function should emit some event and this event should be declared in this interface.

Client Comment *These module functions are wrappers for library functions that emit relevant events.*

```
27 function setInstrumentRegistrationFlag(address instrumentAddress,
   ↪ bool isRegistered) external;
```

CVF-242 INFO

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

Recommendation The type for the “quoteCollateral” argument should be more specific.

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

```
30 function registerMarket(address quoteCollateral, string memory name)
   ↪  external returns (uint128 newMarketId);
```

CVF-243 INFO

- **Category** Suboptimal
- **Source** IInstrumentModule.sol

Recommendation This function should emit some event and this event should be declared in this interface.

Client Comment *These module functions are wrappers for library functions that emit relevant events.*

```
30 function registerMarket(address quoteCollateral, string memory name)
   ↪  external returns (uint128 newMarketId);
```

CVF-244 INFO

- **Category** Suboptimal
- **Source** IExchangeManagerModule.sol

Recommendation This function should emit some event and this event should be declared in this interface.

Client Comment *These module functions are wrappers for library functions that emit relevant events.*

```
25 function registerExchange(uint128 exchangeFeeCollectorAccountId)
   ↪ external returns (uint128 exchangeId);
```

CVF-245 INFO

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

Recommendation The type for the “collateral” argument should be more specific.

Client Comment *See CVF-89.*

```
27 function getCollateralPoolBalance(uint128 collateralPoolId, address
   ↪ collateral) external view returns (uint256);
```

CVF-246 INFO

- **Category** Procedural
- **Source** ICollateralPoolModule.sol

Recommendation These functions should emit some events and these events should be declared in this interface.

Client Comment *These module functions are wrappers for library functions that emit relevant events.*

```
42 function transferCollateralPoolOwnership(uint128 collateralPoolId,
   ↪ address newOwner) external;
```

```
61 function mergeCollateralPools(uint128 parentCollateralPoolId,
   ↪ uint128 childCollateralPoolId) external;
```

```
73 function setCollateralPoolLimits(uint128 collateralPoolId,
   ↪ LimitConfig memory limits) external;
```

CVF-247 INFO

- **Category** Suboptimal
- **Source** IProtocolConfigurationModule.sol

Recommendation This function should emit some events and this event should be declared in this interface.

Client Comment *These module functions are wrappers for library functions that emit relevant events.*

```
28 function configureProtocol(ProtocolConfiguration.Data memory config)
     ↪  external;
```

CVF-248 INFO

- **Category** Procedural
- **Source** IRiskConfigurationModule.sol

Description We didn't review this file.

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

CVF-249 INFO

- **Category** Suboptimal
- **Source** IRiskConfigurationModule.sol

Recommendation These functions should emit some events and these events should be declared in this interface.

Client Comment *These module functions are wrappers for library functions that emit relevant events.*

```
31 function configureRiskMultipliers(uint128 collateralPoolId,
   ↪ RiskMultipliers memory config) external;

43 function configureLiquidation(uint128 collateralPoolId,
   ↪ LiquidationConfig memory config) external;

58 function createRiskMatrix(uint128 collateralPoolId, SD59x18[][]
   ↪ memory values) external returns (uint128 blockId);

68 function setBackstopLPConfig(uint128 collateralPoolId,
   ↪ BackstopLPConfig memory config) external;
```

CVF-250 INFO

- **Category** Procedural
- **Source** CoreProxy.sol

Description We didn't review this file.

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

CVF-251 INFO

- **Category** Procedural
- **Source** CoreProxy.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