File	Lines	Code	CVF	Severity	Category	Description	Recommendation

MarketUtils.sol	156, 181	`156    // the value of a market's liquidity pool is the worth of the liquidity provider tokens in the pool + pending trader pnl`
		`181         return Calc.sum(value, pnl);`
			1	Critical	Flaw	A trader's profit results in a pool's loss, so the pending trader PnL should be subtracted rather than added.	

SwapUtils.sol	84	`84        PricingUtils.transferFees(`	2	Major	Suboptimal	Transferring fees on every atomic swap is inefficient.	Consider accumulating fees in the pool and allowing the fee receiver to take them later.

ExchangeRouter.sol	125–127	`125         address weth = EthUtils.weth(dataStore);`
		`126         IWETH(weth).deposit{value: msg.value}();`
		`127         IERC20(weth).safeTransfer(address(receiver), msg.value);`
			3	Major	Suboptimal	This code should be executed only when msg.value is not zero.	

PositionUtils.sol	53–59, 61–63	`53        if (totalPositionPnl > 0) {`
		`54            sizeDeltaInTokens = position.sizeInTokens * sizeDeltaUsd / position.sizeInUsd;`
		`55        } else {`
		`56            uint256 nextSizeInUsd = position.sizeInUsd - sizeDeltaUsd;`
		`57            uint256 nextSizeInTokens = position.sizeInTokens * nextSizeInUsd / position.sizeInUsd;`
		`58            sizeDeltaInTokens = position.sizeInTokens - nextSizeInTokens;`
		`59        }`
		`61        if (position.sizeInUsd == sizeDeltaUsd) {`
		`62            sizeDeltaInTokens = position.sizeInTokens;`
		`63        }`
			4	Major	Readability		This should be reordered as:
							if (position.sizeInUsd == sizeDeltaUsd) {...}
							else if (totalPositionPnl > 0) {...}
							else {...}

PositionStore.sol	20	`20        accountPositionKeys[account].add(key);`	5	Major	Unclear behavior	This allows associating the same key with several accounts.	Consider either forbidding to add the same key twice, or removing the key from the previous account like this:
							if (!positionKeys.add (key)) {
							  accountPositionKeys [positions [key]. account].remove (key);
							}

PositionStore.sol	26	`26        accountPositionKeys[account].remove(key);`	6	Major	Unclear behavior	In case of incorrect “account” value, this line will not be able to remove the account to key association.	Consider obtaining the account from the “positions” mapping rather than accepting as an argument.
IncreasePositionUtils.sol	66	`66        uint256 sizeDeltaInTokens = params.order.sizeDeltaUsd() / prices.indexTokenPrice;`	7	Major	Flaw	Division by a scaled price could lead to precision degradation for cheap tokens with lots of decimals.	Consider scaling up the denominator and then dividing by an unscaled price.
IncreasePositionUtils.sol	140	`140         PricingUtils.transferFees(`	8	Major	Suboptimal	Sending out fees on every position change is inefficient.	Consider accumulating fees in the pool and allowing the fee receiver to take them later.

MarketUtils.sol	491–492	`491         totalBorrowing -= prevPositionSizeInUsd * prevPositionBorrowingFactor;`
		`492         totalBorrowing += nextPositionSizeInUsd * nextPositionBorrowingFactor;`
			9	Major	Overflow/Underflow	Phantom underflow is possible here.	Consider doing the subtraction after the addition.

MarketStore.sol	25	`25        marketTokens.remove(marketToken);`	10	Major	Unclear behavior	What will happen with deposits and orders in case if the market will be removed? It looks like all functionality related to createDeposit, withdrawDeposit, liquidation and orders management will be broken.	Consider adding a document in which case market could be removed from the MarketStore and what will happen in this case. Consider the usage of "emergency" mode on removed markets allowing only withdrawal.
GasUtils.sol	57	`57        bool success = payable(user).send(refundFeeForUser);`	11	Major	Flaw	In case of an unsuccessful transfer, ether will be accumulated at the contract's balance.	Consider implementing some way for the user to extract this ether later. For example, maintain a mapping of pending ether per user.
GasUtils.sol	63	`63        uint256 minExecutionFee = gasLimit * tx.gasprice;`	12	Major	Flaw	Transaction gas price could be manipulated by a message sender.	Consider using a reliable gas price oracle instead.

WithdrawalUtils.sol	106, 204	`106         require(withdrawal.account != address(0), "WithdrawalUtils: empty withdrawal");`
		`204         require(withdrawal.account != address(0), "WithdrawalUtils: empty withdrawal");`
			13	Major	Unclear behavior		There is no restriction for account in createWithdrawal, consider adding additional check inside createWithdrawal (or use msg.sender)

EnumerableValues.sol	17–19, 30–32, 43–45	`17        for (uint256 i = start; i < end; i++) {`
		`18            items[i - start] = set.at(i);`
		`19        }`
		`30        for (uint256 i = start; i < end; i++) {`
		`31            items[i - start] = set.at(i);`
		`32        }`
		`43        for (uint256 i = start; i < end; i++) {`
		`44            items[i - start] = set.at(i);`
		`45        }`
			14	Major	Suboptimal	Copying array elements one by one is inefficient.	Consider using the “identity” precompile to copy a memory range directly from the array underneath the set.

SwapPricingUtils.sol	126–128	`126         uint256 spreadFactor = dataStore.getUint(Keys.swapSpreadFactorKey(marketToken));`
		`127         uint256 feeFactor = dataStore.getUint(Keys.swapFeeFactorKey(marketToken));`
		`128         uint256 feeReceiverFactor = dataStore.getUint(feeReceiverFactorKey);`
			15	Major	Suboptimal		Three parameters here seem redundant. Two would be enough.

PositionPricingUtils.sol	62	`62        bool isSameSideRebalance = openInterestParams.longOpenInterest <= openInterestParams.shortOpenInterest == openInterestParams.nextLongOpenInterest <= openInterestParams.nextShortOpenInterest;`	16	Major	Unclear behavior	This formula is asymmetric. For example, it allows (long < short, nextLong = nextShort), but doesn't allow (long > short, nextLong = nextShort).	Consider handling the equality cases properly.
PositionPricingUtils.sol	90	`90    function getNextOpenInterest(`	17	Major	Unclear behavior	This function always increases an open interest (either long or short) but never decreases it.	Consider decreasing the other side open interest when possible.

PositionPricingUtils.sol	139–141	`139         uint256 feeFactor = dataStore.getUint(Keys.positionFeeFactorKey(position.market));`
		`140         uint256 feeReceiverFactor = dataStore.getUint(feeReceiverFactorKey);`
		`141         uint256 spreadFactor = dataStore.getUint(Keys.positionSpreadFactorKey(position.market));`
			18	Major	Suboptimal		Three parameters seem redundant. Two would be enough: the fee percentage that goes to the fee receiver, and the spread percentage that goes to the pool.

File	Lines	Code	CVF	Severity	Category	Description	Recommendation
OrderUtils.sol	72	`72        address account,`	19	Major	Unclear behavior		As zero account is used as a marker of a non-existing order, consider explicitly requiring the account to be not zero.

OrderUtils.sol	162–164, 168–170, 174–175, 179–180, 184–185, 189–191, 207	`162        return orderType == Order.OrderType.MarketSwap ||`
		`163               orderType == Order.OrderType.MarketIncrease ||`
		`164               orderType == Order.OrderType.MarketDecrease;`
		`168        return orderType == Order.OrderType.LimitSwap ||`
		`169               orderType == Order.OrderType.LimitIncrease ||`
		`170               orderType == Order.OrderType.LimitDecrease;`
		`174        return orderType == Order.OrderType.MarketSwap ||`
		`175               orderType == Order.OrderType.LimitSwap;`
		`179        return orderType == Order.OrderType.MarketIncrease ||`
		`180               orderType == Order.OrderType.LimitIncrease;`
		`184        return orderType == Order.OrderType.MarketIncrease ||`
		`185               orderType == Order.OrderType.LimitIncrease;`
		`189        return orderType == Order.OrderType.MarketDecrease ||`
		`190               orderType == Order.OrderType.LimitDecrease ||`
		`191               orderType == Order.OrderType.StopLossDecrease;`
		`207        if (orderType == Order.OrderType.MarketIncrease || orderType == Order.OrderType.`
		`MarketDecrease) {`
			20	Major	Suboptimal		This could be optimized as:
							return (1 << uint256 (OrderType. MarketSwap) | 1 << uint256 (OrderType. MarketIncrease) | 1 << uint256 (OrderType.MarketDecrease)) & 1 << uint256 (orderType) != 0;
							Note, that the expression to the left of “&” is constant and thus will be computed at compile time.

Timelock.sol	43	`43    function fastSetUints(DataStore dataStore, string[] memory prefixes, bytes[] memory data,`
		`uint256[] memory values) external onlyAdmin {`
			21	Major	Flaw	There is no length check for the arguments. If the “prefix” array length is smaller than the lengths of the other arrays, then remaining parts of the other arrays will be silently ignored.	Consider adding appropriate length checks.

RoleStore.sol	19–20	`19        roles.add(key);`
		`20        roleMembers[key].add(account);`
			22	Major	Suboptimal		It would be more efficient to do like this:
							if (roleMembers [key].add (account))
							roles.add (key);

SwapOrderUtils.sol	12	`12    function processOrder(OrderUtils.ExecuteOrderParams memory params) external {`	23	Major	Suboptimal	This function should emit some event.	
SwapOrderUtils.sol	17	`17        address firstMarket = params.order.swapPath()[0];`	24	Major	Flaw	There is no explicit check to ensure that the swap path is not empty.	Consider adding such a check.
SwapOrderUtils.sol	41	`41            address(0)`	25	Major	Suboptimal		This should be “order.account()” to make the separate “transferOut” call unnecessary.
MarketFactory.sol	19	`19    function createMarket(`	26	Major	Suboptimal	This function should emit some event.	

DecreasePositionUtils.sol	73–75	`73        if (values.remainingCollateralAmount < 0) {`
		`74            revert("Insufficient collateral");`
		`75        }`
			27	Major	Procedural		This check should be moved into the “processCollateral” function. Having it here is error-prone.

DecreasePositionUtils.sol	196, 219	`196            remainingCollateralAmount += values.realizedPnlAmount;`
		`219            remainingCollateralAmount += fees.totalNetCostAmount;`
			28	Major	Flaw	The remaining collateral may go negative here.	Consider adding an explicit check to prevent this.

Precision.sol	16, 20	`16        return amount * factor / FLOAT_PRECISION;`
		`20        return amount.toInt256() * factor / FLOAT_PRECISION.toInt256();`
			29	Major	Overflow/Underflow	Phantom overflow is possible here, i.e. a situation when the final calculation result would fit into the destination type while some intermediary calculation overflows.	Consider using the “muldiv” function as described here: https://xn--2-umb.com/21/muldiv/

Role.sol	6–11	`6    bytes32 public constant CONTROLLER = keccak256("CONTROLLER");`
		`7    bytes32 public constant ROUTER_PLUGIN = keccak256("ROUTER_PLUGIN");`
		`8    bytes32 public constant MARKET_KEEPER = keccak256("MARKET_KEEPER");`
		`9    bytes32 public constant ORDER_KEEPER = keccak256("ORDER_KEEPER");`
		`10    bytes32 public constant PRICING_KEEPER = keccak256("PRICING_KEEPER");`
		`11    bytes32 public constant LIQUIDATION_KEEPER = keccak256("LIQUIDATION_KEEPER");`
			30	Major	Procedural	Public constants don't make much sense in a library.	Consider changing the acces level to “internal”.

FeeReceiver.sol	8	`8    function notifyFeeReceived(bytes32 key, address token, uint256 amount) external {`	31	Major	Suboptimal	The function is declared as external so anyone can call it and trigger backend on false event.	Consider restricting the caller or include emitting inside the usage code.
WithdrawalHandler.sol	22	`22contract WithdrawalHandler is RoleModule, ReentrancyGuard, OracleModule {`	32	Major	Unclear behavior	There is no separate function to cancel a withdrawal.	Consider adding such a function.

WithdrawalHandler.sol	49, 82	`49    function createWithdrawal(`
		`82    function executeWithdrawal(`
			33	Major	Suboptimal	These functions should emit some events.	

DepositUtils.sol	78	`78            params.account,`	34	Major	Suboptimal		As zero account is used as as a marker of a non-existing deposit, consider explicitly requiring the account to bo non-zero.
LiquidationHandler.sol	46	`46    function liquidatePosition(`	35	Major	Suboptimal	This function should emit some event.	

DataStore.sol	52	`52    function decrementUint(bytes32 key, int256 value) external onlyController returns (int256)`
		`{`
			36	Major	Bad naming		Confused naming may lead to misusing the function, consider renaming to decrementInt

SwapPricingUtils.sol	68	`68        bool isSameSideRebalance = poolParams.poolUsdForTokenA <= poolParams.poolUsdForTokenB`
		`== poolParams.nextPoolUsdForTokenA <= poolParams.nextPoolUsdForTokenB;`
			37	Major	Unclear behavior	This formula is asymmetric. For example, it allows (A = B, nextA < nextB) but doesn't allow (A = B, nextA > nextB).	Consider handling the A = B and nextA = nextB cases properly.

RoleStore.sol	23	`23    function revokeRole(address account, bytes32 key) external onlyGov {`	220	Major	Flaw	This function doesn't remove roles with no members from the “roles” set.	Consider refactoring like this:
							EnumerableSet.Bytes32Set storage members = roleMembers [key];
							if (members.remove (account) && members.length () == 0)
							roles.remove (key);

File	Lines	Code	CVF	Severity	Category	Description	Recommendation
PositionUtils.sol	75	`75        bytes32 key = keccak256(abi.encodePacked(account, market, collateralToken, isLong));`	38	Moderate	Unclear behavior	Here should be some unique prefix hashed to guaranteed key uniqueness.	
Oracle.sol	79	`79        SALT = keccak256(abi.encodePacked(block.chainid, "xget-oracle-v1"));`	39	Moderate	Flaw	block.chainid may change in future or in forks. The "freezing" the chainid allows replay attack on forks.	Consider using CACHED chain_id as it is done in https://github.com/fractional-company/contracts/blob/master/src/OpenZeppelin/drafts/EIP712.sol . Also check the discusssion here - https://forum.openzeppelin.com/t/why-does-domainseparatorv4-in-eip712-hashtypeddatav4-in-erc20permit-check-for-address-and-chainid/25359

MarketUtils.sol	199, 345, 541	`199         int256 openInterestValue = (openInterestInTokens * indexTokenPrice).toInt256();`
		`345              reservedUsd = openInterestInTokens * prices.indexTokenPrice;`
		`541         uint256 poolUsd = poolAmount * poolTokenPrice;`
			40	Moderate	Unclear behavior	Here a price is already scaled according to the index token decimals. This could lead to precision degradation for cheap tokens with lots of decimals.	Consider scaling a product, rather than a price.

MarketStore.sol	33	`33        return markets[marketToken];`	41	Moderate	Flaw		Consider checking if market is empty or not, to be sure that specific market was not already removed.
GasUtils.sol	39	`39        uint256 executionFeeForKeeper = adjustGasLimit(dataStore, gasUsed) * tx.gasprice;`	42	Moderate	Flaw	Transaction gas price could be manipulated by a message sender. When a message sender knows that gas will be refunded, he could set a gas price that is higher than the current market price, and the user will pay extra costs.	Consider using a reliable gas price oracle such as chainlink, instead of just the gas price of the current transaction.

EnumerableValues.sol	16, 29, 42	`16        bytes32[] memory items = new bytes32[](end - start);`
		`29        address[] memory items = new address[](end - start);`
		`42        uint256[] memory items = new uint256[](end - start);`
			43	Moderate	Unclear behavior	This line throws in case start > end. The caller cannot efficiently prevent this, as the "end" value could be adjusted in the previous line.	Consider returning an empty array in case start >= end here.

OrderUtils.sol	307	`307             orderType == Order.OrderType.LimitIncrease ||`	44	Moderate	Unclear behavior	This conditions is always false here, as the "orderType == Order.OrderType. LimitIncrease" case is handled by tyhe previous conditional statement.	

OracleModule.sol	18–19	`18            emit OracleError(reason);`
		`19            revert(Keys.ORACLE_ERROR);`
			45	Moderate	Procedural	Emitting an event before reverting a transaction doesn't make sense, as the event will be reverted as well.	Consider returning instead of reverting.

OrderHandler.sol	130	`130    function cancelOrder(bytes32 key) external {`	46	Moderate	Suboptimal	The owner of an order may frontrun order execution transactions from keepers with order CANCEL requests, thus making executions to fail and keepers to loose money.	

DepositUtils.sol	113–114, 126–127	`113         uint256 longTokenUsd = deposit.longTokenAmount * longTokenPrice;`
		`114         uint256 shortTokenUsd = deposit.shortTokenAmount * shortTokenPrice;`
		`126                 (deposit.longTokenAmount * longTokenPrice).toInt256(),`
		`127                 (deposit.shortTokenAmount * shortTokenPrice).toInt256()`
			47	Moderate	Suboptimal	For cheap tokens with lots of decimals, using a scaled price could lead to precision degradation.	Consider multiplying by an unscaled price and scaling the product.

File	Lines	Code	CVF	Severity	Category	Description	Recommendation
Keys.sol	93–96, 101–104, 132–135, 139–142, 146–149, 153–156, 160–164, 168–172, 176–180, 184–188, 192–195, 199–203, 207–210, 214–217, 221–224, 228–232, 236–239, 244–247, 251–254, 258–261, 266–270, 275–279, 284–289, 294–298, 303–307, 311–315, 319–322, 326–330, 334–337, 341–345, 349–353, 357–361, 365–369, 373–376, 380–383, 387–390	93 return keccak256(abi.encodePacked(94 DEPOSIT_GAS_LIMIT, 95 singleToken 96)); 101 return keccak256(abi.encodePacked(102 WITHDRAWAL_GAS_LIMIT, 103 singleToken 104)); 132 return keccak256(abi.encodePacked(133 CREATE_DEPOSIT_FEATURE, 134 module 135)); 139 return keccak256(abi.encodePacked(140 EXECUTE_DEPOSIT_FEATURE, 141 module 142)); 146 return keccak256(abi.encodePacked(147 CREATE_WITHDRAWAL_FEATURE, 148 module 149)); 153 return keccak256(abi.encodePacked(154 EXECUTE_WITHDRAWAL_FEATURE, 155 module 156)); 160 return keccak256(abi.encodePacked(161 CREATE_ORDER_FEATURE, 162 module, 163 orderType 164)); 168 return keccak256(abi.encodePacked(169 EXECUTE_ORDER_FEATURE, 170 module, 171 orderType 172)); 176 return keccak256(abi.encodePacked(177 UPDATE_ORDER_FEATURE, 178 module, 179 orderType 180)); 184 return keccak256(abi.encodePacked(185 CANCEL_ORDER_FEATURE, 186 module, 187 orderType 188)); 192 return keccak256(abi.encodePacked(193 LIQUIDATE_POSITION_FEATURE, 194 module 195)); 199 return keccak256(abi.encodePacked(200 POSITION_IMPACT_FACTOR, 201 market, 202 isPositive 203)); 207 return keccak256(abi.encodePacked(208 POSITION_IMPACT_EXPONENT_FACTOR, 209 market 210)); 214 return keccak256(abi.encodePacked(215 POSITION_SPREAD_FACTOR, 216 market 217)); 221 return keccak256(abi.encodePacked(222 POSITION_FEE_FACTOR, 223 market 224)); 228 return keccak256(abi.encodePacked(229 SWAP_IMPACT_FACTOR, 230 market, 231 isPositive 232)); 236 return keccak256(abi.encodePacked(237 SWAP_IMPACT_EXPONENT_FACTOR, 238 market 239)); 244 return keccak256(abi.encodePacked(245 SWAP_SPREAD_FACTOR, 246 market 247)); 251 return keccak256(abi.encodePacked(252 SWAP_FEE_FACTOR, 253 market 254)); 258 return keccak256(abi.encodePacked(259 ORACLE_PRECISION, 260 token 261)); 266 return keccak256(abi.encodePacked(267 OPEN_INTEREST, 268 market, 269 isLong 270)); 275 return keccak256(abi.encodePacked(276 OPEN_INTEREST_IN_TOKENS	48	Moderate	Suboptimal	encodePacked function is not injective, so that it is possible that different inputs concatenate to the same output of the function.	In order to avoid hash collisions consider ensuring that this situation is impossible by making all first arguments prefix free, i.e. no one can be a prefix of another one. One way to ensure that is to append a symbol that would never occur, or prehash the constants.

File	Lines	Code	CVF	Severity	Category	Description	Recommendation
WithdrawalStore.sol	12	`12 mapping(bytes32 => Withdrawal.Props) public withdrawals;`	49	Minor	Unclear behavior	It's not clear why to use hash of uint256 as a key.	Consider the usage of just uint256 nonce as a key to increase readability and avoid redundant hashing.
WithdrawalStore.sol	15	`15 constructor(RoleStore _roleStore) StrictBank(_roleStore) {}`	50	Minor	Procedural		It is a good practice to put a comment into an empty block to explain why the block is empty.
WithdrawalStore.sol	17	`17 function set(bytes32 key, Withdrawal.Props memory withdrawal) external onlyController {`	51	Minor	Bad naming	The name "set" is usually associated with setting a flag or an atomic value. Here a value for a key is set. Such functions are usually named "put".	Consider also the name "store".
WithdrawalStore.sol	19	`19 withdrawalKeys.add(key);`	52	Minor	Suboptimal	An event should be emitted here	
Array.sol	6	`6 function get(bytes32[] memory arr, uint256 index) internal pure returns (bytes32) {`	53	Minor	Suboptimal		This function would be more useful if it would accept the default value as an argument.
Array.sol	6	`6 function get(bytes32[] memory arr, uint256 index) internal pure returns (bytes32) {`	54	Minor	Bad naming	The name is too generic.	Consider making it more specific to emphasize the ability of this function to handle invalid indexes.

Array.sol	8, 16, 21	`8 return arr[index];`
		`16 arr[index] = value;`
		`21 newArr[index] = value;`
			55	Minor	Suboptimal		Solidity arrays have length check inside, consider using unsafe reading/setting array slot like in https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.8.0-rc.2/contracts/utils/Arrays.sol#L87

Array.sol	15-18	`15 if (index < arr.length) {`
		`16 arr[index] = value;`
		`17 return arr;`
		`18 }`
			56	Minor	Bad naming	Confusing function name as no copy is created.	Consider renaming

Array.sol	20	`20 bytes32[] memory newArr = createResized(arr, index + 1);`	57	Minor	Suboptimal	Increasing the array size to index + 1 is suboptimal, as it may lead to O(n^2) complexity for sequential writes.	Consider increasing the array size to index * 3 / 2 + 1 or something like this.

Array.sol	27-29	`27 if (length <= arr.length) {`
		`28 return arr;`
		`29 }`
			58	Minor	Documentation		Nothing is created here, consider documenting it.

Array.sol	31	`31 bytes32[] memory newArr = new bytes32[](length);`	59	Minor	Suboptimal	In a quite common case when arr + arr.length equals to the free memory pointer, it is possible to resize the array without copying.	Consider implementing such logic.

Array.sol	33–35	`33 for (uint256 i = 0; i < arr.length; i++) {`
		`34 newArr[i] = arr[i];`
		`35 }`
			60	Minor	Suboptimal	Copying array elements only by one is suboptimal.	Consider using the identity precompile.

Array.sol	33, 41, 51	`33 for (uint256 i = 0; i < arr.length; i++) {`
		`41 for (uint256 i = 0; i < arr.length; i++) {`
		`51 for (uint256 i = 0; i < arr.length; i++) {`
			61	Minor	Suboptimal		Use unchecked declaration for i++ to save gas.

Array.sol	60	`60 function getMedian(uint256[] memory arr) internal pure returns (uint256) {`	62	Minor	Unclear behavior	This function works correctly only for sorted arrays.	Consider reflecting this fact in the function name or in a documentation comment.

Array.sol	60-66	`60 function getMedian(uint256[] memory arr) internal pure returns (uint256) {`
		`61 if (arr.length % 2 == 1) {`
		`62 return arr[arr.length / 2];`
		`63 }`
		`64`
		`65 return (arr[arr.length / 2] + arr[arr.length / 2 - 1]) / 2;`
		`66 }`
			63	Minor	Flaw		The function will fail if length=0, consider adding a special check for this case.

Array.sol	65	`65 return (arr[arr.length / 2] + arr[arr.length / 2 - 1]) / 2;`	64	Minor	Suboptimal		Right shift would be more efficient that division.
Array.sol	65	`65 return (arr[arr.length / 2] + arr[arr.length / 2 - 1]) / 2;`	65	Minor	Suboptimal	The expression "arr.length / 2" is calculated twice.	Consider calculating once and reusing.
Array.sol	65	`65 return (arr[arr.length / 2] + arr[arr.length / 2 - 1]) / 2;`	66	Minor	Overflow/Underflow	Phantom overflow is possible here, i.e. a situation when the final calculation result would fit into the destination type, while some intermediary calculation overflows.	

SwapUtils.sol	12, 23	`12 struct SwapParams {`
		`23 struct _SwapParams {`
			67	Minor	Suboptimal	Having two structs with very similar names is confusing.	Consider using more distinct names.

SwapUtils.sol	16, 25, 31	`16 address tokenIn;`
		`25 address tokenIn;`
		`31 address tokenOut;`
			68	Minor	Bad datatype		The type of these fields could be more specific.

SwapUtils.sol	39	`39 // returns tokenOut, outputAmount`	69	Minor	Suboptimal		Consider declaring return values in the function declaration or use NatSpec.

SwapUtils.sol	41–42	`41 address tokenOut = params.tokenIn;`
		`42 uint256 outputAmount = params.amountIn;`
			70	Minor	Bad naming	These variable names are confusing, as they used as both, input and output.	Consider renaming to just "token" and "amount", or "currentToken" and "currentAmount".

File	Lines	Code	CVF	Severity	Category	Description	Recommendation

SwapUtils.sol	44-61	44 for (uint256 i = 0; i < params.markets.length; i++) {
		45 Market.Props memory market = params.markets[i];
		46 uint256 nextIndex = i + 1;
		47 address receiver;
		48 if (nextIndex < params.markets.length) {
		49 receiver = params.markets[nextIndex].marketToken;
		50 } else {
		51 receiver = params.receiver;
		52 }
		53
		54 _SwapParams memory _params = _SwapParams(
		55 market,
		56 tokenOut,
		57 outputAmount,
		58 receiver
		59);
		60 (tokenOut, outputAmount) = _swap(params, _params);
		61 }
			71	Minor	Suboptimal		Consider using unchecked declaration to save gas.

SwapUtils.sol	99–100	99 (fees.amountAfterFees * cache.tokenInPrice).toInt256(), 100 -(fees.amountAfterFees * cache.tokenInPrice).toInt256()	72	Minor	Suboptimal	The expression "fees.amountAfterFees * cache.tokenInPrice" is calculated twice.	Consider calculating once and reusing.

SwapUtils.sol	105, 137	105 cache.amountOut = cache.amountIn * cache.tokenInPrice / cache.tokenOutPrice;
		137 cache.amountOut = cache.amountIn * cache.tokenInPrice / cache.tokenOutPrice;
			73	Minor	Overflow/Underflow	Phantom overflow is possible here.	Consider using the "muldiv" function.

ExchangeRouter.sol	19–26	19 Router public router;
		20 DataStore public dataStore;
		21 DepositHandler public depositHandler;
		22 WithdrawalHandler public withdrawalHandler;
		23 OrderHandler public orderHandler;
		24 DepositStore public depositStore;
		25 WithdrawalStore public withdrawalStore;
		26 OrderStore public orderStore;
			74	Minor	Procedural		These variables should be declared as immutable.

ExchangeRouter.sol	60, 90, 110	60 address account = msg.sender;
		90 address account = msg.sender;
		110 address account = msg.sender;
			75	Minor	Suboptimal		This variable is redundant, as "msg. sender" is cheaper to read than a local variable.

ExchangeRouter.sol	94-103	94 return withdrawalHandler.createWithdrawal(95 account, 96 market, 97 marketTokensLongAmount, 98 marketTokensShortAmount, 99 minLongTokenAmount, 100 minShortTokenAmount, 101 hasCollateralInETH, 102 executionFee 103);	76	Minor	Procedural		Consider using named arguments
PositionUtils.sol	28	28) internal pure returns (int256, uint256) {	77	Minor	Documentation	The semantics of the returned values is unclear.	Consider giving descriptive names to the returned values and/or adding a documentation comment.
PositionUtils.sol	38	38 // returns (positionPnlUsd, sizeDeltaInTokens)	78	Minor	Bad naming		Consider giving these names to the returned values, rather then specifying them in a comment.
PositionUtils.sol	38	38 // returns (positionPnlUsd, sizeDeltaInTokens)	79	Minor	Bad naming	The name "positionPnlUsd" is confusing, as one could thing that this is the current USD-denominated PnL for the whole position, while actually this is not the case.	Consider renaming.

PositionUtils.sol	54, 65, 71, 154	54 sizeDeltaInTokens = position.sizeInTokens * sizeDeltaUsd / position.sizeInUsd;
		65 int256 positionPnlUsd = totalPositionPnl * sizeDeltaInTokens.toInt256() / position.
		sizeInTokens.toInt256();
		71 return sizeInTokens * sizeDeltaUsd / sizeInUsd;
		154 if (position.sizeInUsd * Precision.FLOAT_PRECISION / remainingCollateralUsd.
		toUint256() > maxLeverage) {
			80	Minor	Overflow/Underflow	Phantom overflow is possible here.	Consider using the "muldiv" function.

PositionUtils.sol	56–58	56 uint256 nextSizeInUsd = position.sizeInUsd - sizeDeltaUsd;
		57 uint256 nextSizeInTokens = position.sizeInTokens * nextSizeInUsd / position.
		sizeInUsd;
		58 sizeDeltaInTokens = position.sizeInTokens - nextSizeInTokens;
			81	Minor	Suboptimal		There are simpler ways to divide with rounding up. For example: (x + y - 1) / y

PositionUtils.sol	65	65 int256 positionPnlUsd = totalPositionPnl * sizeDeltaInTokens.toInt256() / position. sizeInTokens.toInt256();	82	Minor	Documentation	The difference between "totalPositionPnl" and "positionPnlUsd" is unclear.	Consider documenting.
PositionUtils.sol	74	74 function getPositionKey(address account, address market, address collateralToken, bool isLong) internal pure returns (bytes32) {	83	Minor	Bad datatype		The type of the "collateralToken" argument could be more specific.
PositionUtils.sol	80	80 if (position.sizeInUsd == 0 || position.sizeInTokens == 0 || position.collateralAmount == 0) {	84	Minor	Suboptimal		This could be optimized as: if (position.sizeInUsd | position. sizeInTokens | position.collateralAmount == 0)

File	Lines	Code	CVF	Severity	Category	Description	Recommendation
PositionUtils.sol	125-134	125 PositionPricingUtils.GetPositionPricingParams(126 dataStore, 127 market.marketToken, 128 market.longToken, 129 market.shortToken, 130 prices.longTokenPrice, 131 prices.shortTokenPrice, 132 -position.sizeInUsd.toInt256(), 133 position.isLong 134)	85	Minor	Readability		Use named args to avoid argument misusing.

PositionUtils.sol	149–151, 154–156, 158	149 if (remainingCollateralUsd < minCollateralUsd) {
		150 return true;
		151 }
		154 if (position.sizeInUsd * Precision.FLOAT_PRECISION / remainingCollateralUsd.toUint256() > maxLeverage) {
		155 return true;
		156 }
		158 return false;
			86	Minor	Suboptimal		This could be simplified as:
							return remainingCollateralUsd < minCollateralUsd || position.sizeInUsd * precision.FLOAT_PRECISION / remainingCollateralUsd.toUint256() > maxLeverage;

PositionUtils.sol	161	161 function revertUnexpectedPositionState() internal pure {	87	Minor	Procedural	This function is never used and is too simple to be extracted.	Consider removing this function.
PositionStore.sol	16	16 constructor(RoleStore _roleStore) RoleModule(_roleStore) {}	88	Minor	Documentation		It is a good practice to put a comment into an empty bock to explain why the block is empty.
PositionStore.sol	18	18 function set(bytes32 key, address account, Position.Props memory position) external onlyController {	89	Minor	Bad naming		The name "set" is usually associated with setting a flag or an atomic variable. Such functions are usually named "put". Also, consider the name "store".
PositionStore.sol	21	21 positionKeys.add(key);	90	Minor	Suboptimal	An event should be emitted here.	
PositionStore.sol	27	27 positionKeys.remove(key);	91	Minor	Suboptimal	An event should be emitted here.	
PositionStore.sol	50	50 function contains(bytes32 key) public view returns (bool) {	92	Minor	Unclear behavior		For completeness, consider implementing a version of this function with two arguments: "key" and "account".
IncreasePositionUtils.sol	32	32 address collateralToken;	93	Minor	Bad datatype		The type of this field could be more specific.
OrderStore.sol	18	18 constructor(RoleStore _roleStore) StrictBank(_roleStore) {}	94	Minor	Documentation		It is a good practice to put a comment into an empty block to explain why the block is empty.
OrderStore.sol	20	20 function set(bytes32 key, Order.Props memory order) external onlyController {	95	Minor	Bad naming	The name "set" is usually associated with setting a flag or an atomic value, however ere a value for a key is set. Such functions are usually named "put".	Consider also the name "store".
OrderStore.sol	22	22 accountOrderKeys[order.account()].add(key);	96	Minor	Procedural	The previous association for the same key is not removed here, so the same order key could be associated with several accounts.	Consider ether forbidding adding the same order key several times or removing the existing association if any.
OrderStore.sol	23	23 orderKeys.add(key);	97	Minor	Suboptimal	Here an event should be emitted	
OrderStore.sol	26	26 function remove(bytes32 key, address account) external onlyController {	98	Minor	Suboptimal		The "account" argument is redundant as it could be derived as: orders [key].account()
OrderStore.sol	28	28 accountOrderKeys[account].remove(key);	99	Minor	Unclear behavior	In case of incorrect "account" value, this line wouldn't remove the association.	Consider either requiring the association to exist, or deriving an account from the key.
OrderStore.sol	29	29 orderKeys.remove(key);	100	Minor	Unclear behavior	Here an event should be emitted	
IncreaseOrderUtils.sol	12	12 MarketUtils.validateNonEmptyMarket(params.market);	101	Minor	Procedural		This check should be done earlier.
IncreaseOrderUtils.sol	14-23, 52-63	14 (address collateralToken, uint256 collateralDeltaAmount) = SwapUtils.swap(SwapUtils.SwapParams(15 params.dataStore, 16 params.oracle, 17 params.feeReceiver, 18 params.order.initialCollateralToken(), 19 params.order.initialCollateralDeltaAmount(), 20 params.swapPathMarkets, 21 params.order.minOutputAmount(), 22 address(0) 23)); 52 IncreasePositionUtils.IncreasePositionParams(53 params.dataStore, 54 params.positionStore, 55 params.oracle, 56 params.feeReceiver, 57 params.market, 58 params.order, 59 position, 60 positionKey, 61 collateralToken, 62 collateralDeltaAmount 63)	102	Minor	Readability		Consider using named args.

File	Lines	Code	CVF	Severity	Category	Description	Recommendation

IncreaseOrderUtils.sol	31–33	`31 if (position.market != address(0) || position.collateralToken != address(0)) {`
		`32 PositionUtils.revertUnexpectedPositionState();`
		`33 }`
			103	Minor	Suboptimal		This check should be done earlier.

IncreaseOrderUtils.sol	47–49	`47 if (collateralToken != params.market.longToken && collateralToken != params.market.shortToken) {`
		`48 revert("OrderUtils: invalid collateralToken");`
		`49 }`
			104	Minor	Suboptimal		This check should be done earlier.

DecreaseOrderUtils.sol	20–21, 35, 40–41, 43, 46, 62–63, 65–66	`20 params.order.orderType(),`
		`21 params.order.updatedAtBlock(),`
		`35 params.order.sizeDeltaUsd(),`
		`40 if (adjustedSizeDeltaUsd == params.order.sizeDeltaUsd()) {`
		`41 params.orderStore.remove(params.key, params.order.account());`
		`43 params.order.setSizeDeltaUsd(adjustedSizeDeltaUsd);`
		`46 params.orderStore.set(params.key, params.order);`
		`62 params.order.initialCollateralToken(),`
		`63 params.order.initialCollateralDeltaAmount(),`
		`65 params.order.minOutputAmount(),`
		`66 params.order.account()`
			105	Minor	Suboptimal		Here "params.order" could be replaced with just "order".

DecreaseOrderUtils.sol	26-37, 58-67	`26 DecreasePositionUtils.DecreasePositionParams(`
		`27 params.dataStore,`
		`28 params.positionStore,`
		`29 params.oracle,`
		`30 params.feeReceiver,`
		`31 params.market,`
		`32 order,`
		`33 position,`
		`34 positionKey,`
		`35 params.order.sizeDeltaUsd(),`
		`36 forLiquidation`
		`37 )`
		`58 SwapUtils.swap(SwapUtils.SwapParams(`
		`59 params.dataStore,`
		`60 params.oracle,`
		`61 params.feeReceiver,`
		`62 params.order.initialCollateralToken(),`
		`63 params.order.initialCollateralDeltaAmount(),`
		`64 params.swapPathMarkets,`
		`65 params.order.minOutputAmount(),`
		`66 params.order.account()`
		`67 ));`
			106	Minor	Readability		Consider using named args.

DecreaseOrderUtils.sol	49	`49 if (order.swapPath().length == 0) {`	107	Minor	Suboptimal		The actual "swap" call below uses "params. swapPathMarkets" rather than "order. swapPath()", so it would be more logical to use "params.swapPathMarkets" here as well.
OracleUtils.sol	18	`18 uint256 public constant COMPACTED_PRICE_LENGTH = 32;`	108	Minor	Unclear behavior	Is 32 bits really enough for all token prices?	

OracleUtils.sol	32, 53	`32 uint256 slotIndex = index / COMPACTED_PRICES_PER_SLOT;`
		`53 uint256 slotIndex = index / COMPACTED_BLOCK_NUMBERS_PER_SLOT;`
			109	Minor	Suboptimal		Shift would be more efficient here.

OracleUtils.sol	34, 55	`34 uint256 offset = (index - slotIndex * COMPACTED_PRICES_PER_SLOT) * COMPACTED_PRICE_LENGTH;`
		`55 uint256 offset = (index - slotIndex * COMPACTED_BLOCK_NUMBERS_PER_SLOT) * COMPACTED_BLOCK_NUMBER_LENGTH;`
			110	Minor	Suboptimal		Remainder and shift would be more efficient here.

OracleStore.sol	13	`13 constructor(RoleStore _roleStore) RoleModule(_roleStore) {}`	111	Minor	Procedural		It is a good practice to put a comment into an empty block to explain why the block is empty.

OracleStore.sol	16, 20	`16 signers.add(account);`
		`20 signers.remove(account);`
			112	Minor	Unclear behavior	An event should be emitted here.	

Oracle.sol	26	`26 struct _SetPricesCache {`	113	Minor	Bad naming	Starting a structure name with an underscore ("_") looks odd.	
Oracle.sol	30	`30 bytes32 blockHash;`	114	Minor	Suboptimal	Redundant data since we already know the blocknumber.	
Oracle.sol	31	`31 address token;`	115	Minor	Bad datatype		The type of this field could be more specific.
Oracle.sol	32	`32 uint256 prevPrice;`	116	Minor	Documentation	The number format of this field is unclear.	Consider documenting.
Oracle.sol	41	`41 uint256 public constant MAX_SIGNERS = 256 / SIGNER_INDEX_LENGTH - 1;`	117	Minor	Documentation	This equals to 15 which makes Oracle system not-scaleable for hundreds of signers.	Consider adding a comment why 15 signers is enough.
Oracle.sol	42	`42 // signer indexes are recorded in a signerIndexFlags uint256 value to check for uniqueness`	118	Minor	Documentation	In fact the MAX value is 255, 256 is already incorrect.	Consider using 255.
Oracle.sol	45	`45 OracleStore public oracleStore;`	119	Minor	Suboptimal		This variable should be declared as "Immutable".
Oracle.sol	50	`50 EnumerableSet.AddressSet internal tempTokens;`	120	Minor	Unclear behavior	It's not clear why is it neccessary to reset prices in storage.	Consider doing that in memory scope.

File	Lines	Code	CVF	Severity	Category	Description	Recommendation

Oracle.sol	54-56	`54    // the second occurrence will be stored in secondaryPrices`
		`55    mapping(address => uint256) public primaryPrices;`
		`56    mapping(address => uint256) public secondaryPrices;`
			121	Minor	Bad naming	In fact secondary price mapping does not hold second received price but it holds the last price.	Consider renaming to firstPrices, lastPrices.

Oracle.sol	83	`83        require(tempTokens.length() == 0, "Oracle: tempTokens not cleared");`	122	Minor	Suboptimal	String error messages are suboptimal.	Consider using named errors instead.

Oracle.sol	90, 94	`90        if (signers.length < dataStore.getUint(Keys.MIN_ORACLE_SIGNERS)) {`
		`94        if (signers.length > MAX_SIGNERS) {`
			123	Minor	Suboptimal		These checks should be performed with the signes count value before allocating an array.

Oracle.sol	90–91	`90        if (signers.length < dataStore.getUint(Keys.MIN_ORACLE_SIGNERS)) {`
		`91            revert MinOracleSigners(signers.length, dataStore.getUint(Keys.`
		`MIN_ORACLE_SIGNERS));`
			124	Minor	Suboptimal	The expression "dataStore.getUint(Keys. MIN_ORACLE_SIGNERS)" is calculated twice.	Consider calculating once and reusing.

Oracle.sol	131	`131        secondaryPrices[token] = price;`	125	Minor	Unclear behavior	Probably an event should be emitted here.	
Oracle.sol	136	`136        for (uint256 i = 0; i < length; i++) {`	126	Minor	Suboptimal		Consider replacing with "unchecked {i+=1;}"
Oracle.sol	137	`137            address token = tempTokens.at(0);`	127	Minor	Suboptimal		It should be cheaper to remove the last element from the set to avoid storage item swap.
Oracle.sol	210	`210    function getPrecision(DataStore dataStore, address token) public view returns (uint256) {`	128	Minor	Suboptimal	Using different precisions for different tokens make code more complicated.	Consider using the same precision for all tokens.

Oracle.sol	216-220	`216        address[] memory signers,`
		`217        address[] memory tokens,`
		`218        uint256[] memory compactedOracleBlockNumbers,`
		`219        uint256[] memory compactedPrices,`
		`220        bytes[] memory signatures`
			129	Minor	Unclear behavior	There is no validation on alignment of arrays size.	

Oracle.sol	245	`245                cache.blockHash = blockhash(cache.oracleBlockNumber);`	130	Minor	Procedural	Note, that according to the Solidity docs you can only access blockhash for the last 256 blocks.	Consider adding validation of minBlockConfirmations.
Oracle.sol	253	`253                cache.priceAndSignatureIndex = i * signers.length + j;`	131	Minor	Suboptimal	The expression "i * signers.length" is calculated on every loop iteration.	Consider calculating once before the loop.

Oracle.sol	270, 251	`270                prices[j] = price;`
		`251            uint256[] memory prices = new uint256[](signers.length);`
			132	Minor	Suboptimal		If you know the array size in advance you don't need to fill the whole array of numbers, you just need to check ascending of the array and return the middle value.

Oracle.sol	275, 277	`275                secondaryPrices[cache.token] = medianPrice;`
		`277                primaryPrices[cache.token] = medianPrice;`
			133	Minor	Suboptimal		Probably an event should be emitted here.

Oracle.sol	283	`283    // to save costs for tokens with stable prices`	134	Minor	Documentation	It is not clear what "stable prices" mean.	Consider documenting.

Oracle.sol	303	`303            price = price * dataStore.getUint(Keys.priceFeedPrecisionKey(token)) / Precision.`
		`FLOAT_PRECISION;`
			135	Minor	Suboptimal		This line could be simplified using the "applyFactor" function from the "Precision" library.

Oracle.sol	303	`303            price = price * dataStore.getUint(Keys.priceFeedPrecisionKey(token)) / Precision.`
		`FLOAT_PRECISION;`
			136	Minor	Overflow/Underflow	Phantom overflow is possible here.	Consider using the "muldiv" function.

Oracle.sol	307	`307            primaryPrices[token] = price;`	137	Minor	Unclear behavior	Probably an event should be emitted here.	
MarketUtils.sol	39	`39    uint256 public constant MAX_ANNUAL_FUNDING_FACTOR = 1000 * Precision.FLOAT_PRECISION;`	138	Minor	Suboptimal	Annual rates are inefficient.	Consider using per-second rates and converting rates between annual to per-second in UI.

MarketUtils.sol	42–44	`42        uint256 indexTokenPrice;`
		`43        uint256 longTokenPrice;`
		`44        uint256 shortTokenPrice;`
			139	Minor	Documentation	The number format of these fields is unclear.	Consider documenting.

MarketUtils.sol	47, 53, 59, 65	`47    event PoolAmountIncrease(`
		`53    event PoolAmountDecrease(`
		`59    event ImpactPoolAmountIncreased(`
		`65    event ImpactPoolAmountDecreased(`
			140	Minor	Suboptimal	These structures are identical in terms of their fields, and differ only in their names.	Consider using a single structure that could be named "PoolAmountDelta".

MarketUtils.sol	49, 55, 61, 67, 85, 92	`49        address token,`
		`55        address token,`
		`61        address token,`
		`67        address token,`
		`85        address collateralToken,`
		`92        address collateralToken,`
			141	Minor	Bad datatype		The type of these fields could be more specific.

File	Lines	Code	CVF	Severity	Category	Description	Recommendation

MarketUtils.sol	47-67	`47 event PoolAmountIncrease(`
		`48 address market,`
		`49 address token,`
		`50 uint256 amount`
		`51 );`
		`52`
		`53 event PoolAmountDecrease(`
		`54 address market,`
		`55 address token,`
		`56 uint256 amount`
		`57 );`
		`58`
		`59 event ImpactPoolAmountIncreased(`
		`60 address market,`
		`61 address token,`
		`62 uint256 amount`
		`63 );`
		`64`
		`65 event ImpactPoolAmountDecreased(`
		`66 address market,`
		`67 address token,`
			142	Minor	Procedural		Consider indexing two first arguments.

MarketUtils.sol	72–73, 78–79, 84–86, 91–93	`72 address market,`
		`73 bool isLong,`
		`78 address market,`
		`79 bool isLong,`
		`84 address market,`
		`85 address collateralToken,`
		`86 bool isLong,`
		`91 address market,`
		`92 address collateralToken,`
		`93 bool isLong,`
			143	Minor	Suboptimal		These parameters should be indexed.

MarketUtils.sol	73, 79, 86, 93	`73 bool isLong,`
		`79 bool isLong,`
		`86 bool isLong,`
		`93 bool isLong,`
			144	Minor	Suboptimal		It would be more efficient to replace each event declaration with two event declarations: one for long and another for short position.

MarketUtils.sol	104–106	`104 uint256 longTokenPrice,`
		`105 uint256 shortTokenPrice,`
		`106 uint256 indexTokenPrice`
			145	Minor	Unclear behavior	The number format of these arguments in unclear.	

MarketUtils.sol	107	`107 ) internal view returns (uint256) {`	146	Minor	Documentation	The number format of the returned value is unclear.	Consider documenting.
MarketUtils.sol	113	`113 // it may be possible for supply to be zero here`	147	Minor	Documentation	It's not clear why is it possible and why revert division_by_zero is acceptable.	Consider documenting.
MarketUtils.sol	114	`114 return poolValue * Precision.WEI_PRECISION / supply;`	148	Minor	Overflow/Underflow	Phantom overflow is possible here.	Consider using the “muldiv” function.
MarketUtils.sol	114	`114 return poolValue * Precision.WEI_PRECISION / supply;`	149	Minor	Procedural		Scaled division should be extracted into a utility function.
MarketUtils.sol	121	`121 function getOutputToken(address inputToken, Market.Props memory market) internal pure returns (address) {`	150	Minor	Bad datatype		The return type could be more specific.

MarketUtils.sol	125, 136, 139	`125 if (inputToken == market.shortToken) {`
		`136 if (token == market.shortToken) {`
		`139 if (token == market.indexToken) {`
			151	Minor	Readability		Should be “else if” for readability.

MarketUtils.sol	129, 143	`129 revert("MarketUtils: invalid inputToken");`
		`143 revert("MarketUtils: invalid token");`
			152	Minor	Readability		Should be “else revert” for readability.

MarketUtils.sol	132	`132 function getCachedTokenPrice(address token, Market.Props memory market, MarketPrices memory prices) internal pure returns (uint256) {`	153	Minor	Bad datatype		The type of the “token” argument could be more specific.
MarketUtils.sol	143	`143 revert("MarketUtils: invalid token");`	154	Minor	Suboptimal		Consider the usage of custom Error entity.

MarketUtils.sol	150-152	`150 oracle.getSecondaryPrice(market.indexToken),`
		`151 oracle.getPrimaryPrice(market.longToken),`
		`152 oracle.getPrimaryPrice(market.shortToken)`
			155	Minor	Unclear behavior	It is not clear why Secondaty price is mixed with Primary prices, consider adding more explanations in comments.	

MarketUtils.sol	150–152	`150 oracle.getSecondaryPrice(market.indexToken),`
		`151 oracle.getPrimaryPrice(market.longToken),`
		`152 oracle.getPrimaryPrice(market.shortToken)`
			156	Minor	Suboptimal	Here three external calls are performed to the same contract, which is inefficient.	Consider refactoring the oracle API to allow fetching several prices in one call.

MarketUtils.sol	168-169	`168 uint256 longTokenAmount = getPoolAmount(dataStore, market.marketToken, market.longToken);`
		`169 uint256 shortTokenAmount = getPoolAmount(dataStore, market.marketToken, market.shortToken);`
			157	Minor	Documentation	this will not work for deflationary tokens.	Consider documenting.

MarketUtils.sol	193	`193 uint256 openInterestInTokens = getOpenInterestInTokens(dataStore, market, isLong);`	158	Minor	Unclear behavior	This line should be executed only when “openInterest' is not zero.	
MarketUtils.sol	222	`222 if (poolAmount < amount) {`	159	Minor	Suboptimal		This check wouldn't be necessary if the “decrementUint” function would be used instead of “setUint”.

MarketUtils.sol	333, 335, 342	`333 address reserveToken = isLong ? market.longToken : market.shortToken;`
		`335 uint256 reserveTokenPrice = isLong ? prices.longTokenPrice : prices.shortTokenPrice;`
		`342 if (isLong) {`
			160	Minor	Suboptimal		These three conditional executions, that use the same condition, could be merged into a single conditional statement.

File	Lines	Code	CVF	Severity	Category	Description	Recommendation
MarketUtils.sol	382	`382        decreaseImpactPoolAmount(dataStore, market, token, impactAmount);`	161	Minor	Suboptimal		As the current impact amount is already fetched and underflow check is already performed, then it would be cheaper to just calculate and set the new impact amount, rather than decrease it.
MarketUtils.sol	505	`505        if (longOpenInterest + shortOpenInterest == 0) {`	162	Minor	Suboptimal		It would be more efficient to de: longOpenInterest | shortOpenInterest == 0

MarketUtils.sol	509	`509        int256 adjustedFactor = (fundingFactor * diffUsd / (longOpenInterest +`
		`shortOpenInterest) * durationInSeconds).toInt256();`
			163	Minor	Overflow/Underflow	Phantom overflow is possible here.	Consider using the “muldiv” function.

MarketUtils.sol	509	`509        int256 adjustedFactor = (fundingFactor * diffUsd / (longOpenInterest +`
		`shortOpenInterest) * durationInSeconds).toInt256();`
			164	Minor	Suboptimal	Multiplication after division could lead to precision degradation.	Consider performing division at the very end.

MarketUtils.sol	539–540	`539        uint256 poolAmount = getPoolAmount(dataStore, market.marketToken, isLong ? market.`
		`longToken : market.shortToken);`
		`540        uint256 poolTokenPrice = isLong ? prices.longTokenPrice : prices.shortTokenPrice;`
			165	Minor	Suboptimal		There two conditional calculations could be merged into a single conditional statement.

MarketUtils.sol	542	`542        uint256 adjustedFactor = borrowingFactor * openInterest / poolUsd;`	166	Minor	Overflow/Underflow	Phantom overflow is possible here.	Consider using the “muldiv” function.
MarketUtils.sol	557	`557        int256 factor = adjustedFactor * multiplier.toInt256() / divisor.toInt256();`	167	Minor	Overflow/Underflow	Phantom overflow is possible here.	Consider using the “muldiv” function.

MarketUtils.sol	558	`558        int256 maxFactor = (MAX_ANNUAL_FUNDING_FACTOR * durationInSeconds / (365 days)).`
		`toInt256();`
			168	Minor	Bad datatype		The value 365 days” should be a named constant.

MarketUtils.sol	564	`564        return factor;`	169	Minor	Readability		Should be “else return” for readability.

MarketUtils.sol	595, 607	`595        return usdValue * supply / poolValue;`
		`607        return marketTokenAmount * poolValue / supply;`
			170	Minor	Overflow/Underflow	Phantom overflow is possible here.	Consider using the “muldiv” function.

MarketStore.sol	16	`16    constructor(RoleStore _roleStore) RoleModule(_roleStore) {}`	171	Minor	Procedural		It is a good practice to put a comment into an empty block to explain why the block is empty.
MarketStore.sol	18	`18    function set(address marketToken, Market.Props memory market) external onlyController {`	172	Minor	Bad naming	The name “set” is usually associated with setting a flag or an atomic variable, while here a value for a key is set. Such functions are usually named “put”.	Consider also the name “store”.

MarketStore.sol	20, 25	`20        marketTokens.add(marketToken);`
		`25        marketTokens.remove(marketToken);`
			173	Minor	Suboptimal		Consider emitting an event.

GasUtils.sol	21–22	`21    event KeeperExecutionFee(address keeper, uint256 amount);`
		`22    event UserRefundFee(address keeper, uint256 amount, bool success);`
			174	Minor	Procedural		The “keeper” parameters should be indexed.

GasUtils.sol	45	`45        payable(keeper).transfer(executionFeeForKeeper);`	175	Minor	Procedural	Use of the “transfer” function is discouraged.	Consider using the “call” function instead.

GasUtils.sol	88, 96	`88        return dataStore.getUint(Keys.depositGasLimitKey(false));`
		`96        return dataStore.getUint(Keys.withdrawalGasLimitKey(false));`
			176	Minor	Readability		Should be “else return” for readability.

GasUtils.sol	88, 96	`88        return dataStore.getUint(Keys.depositGasLimitKey(false));`
		`96        return dataStore.getUint(Keys.withdrawalGasLimitKey(false));`
			177	Minor	Documentation	What if the parameter is not set?	Consider documenting this case and maybe add additional check.

GasUtils.sol	104, 108	`104        if (OrderUtils.isDecreaseOrder(order.orderType())) {`
		`108        if (OrderUtils.isSwapOrder(order.orderType())) {`
			178	Minor	Readability		Should be “else if” for readability.

GasUtils.sol	112	`112        OrderUtils.revertUnsupportedOrderType();`	179	Minor	Readability		This line should be in an “else” branch for readability.
WithdrawalUtils.sol	35	`35        address weth;`	180	Minor	Bad datatype		The type of this field should be “IWETH”.

WithdrawalUtils.sol	76, 106, 204	`76        require(wethAmount == params.executionFee, "WithdrawalUtils: invalid wethAmount");`
		`106        require(withdrawal.account != address(0), "WithdrawalUtils: empty withdrawal");`
		`204        require(withdrawal.account != address(0), "WithdrawalUtils: empty withdrawal");`
			181	Minor	Suboptimal	String error messages are inefficient.	Consider using named errors instead.

WithdrawalUtils.sol	78, 112	`78        Market.Props memory market = params.marketStore.get(params.market);`
		`112        Market.Props memory market = params.marketStore.get(withdrawal.market);`
			182	Minor	Suboptimal	There is no empty market check.	Consider adding.

File	Lines	Code	CVF	Severity	Category	Description	Recommendation
WithdrawalUtils.sol	80-91, 118-124, 131-140, 144-155, 165-175, 186-192	80 Withdrawal.Props memory withdrawal = Withdrawal.Props(81 params.account, 82 market.marketToken, 83 params.marketTokensLongAmount, 84 params.marketTokensShortAmount, 85 params.minLongTokenAmount, 86 params.minShortTokenAmount, 87 block.number, 88 params.hasCollateralInETH, 89 params.executionFee, 90 new bytes32[](0) 91); 118 cache.poolValue = MarketUtils.getPoolValue(119 params.dataStore, 120 market, 121 longTokenPrice, 122 shortTokenPrice, 123 params.oracle.getPrimaryPrice(market.indexToken) 124); 131 SwapPricingUtils.GetSwapPricingParams(132 params.dataStore, 133 market.marketToken, 134 market.longToken, 135 market.shortToken, 136 longTokenPrice, 137 shortTokenPrice, 138 -(cache.marketTokensLongUsd.toInt256()), 139 -(cache.marketTokensShortUsd.toInt256()) 140) 144 _ExecuteWithdrawalParams memory _params = _ExecuteWithdrawalParams(145 market, 146 withdrawal.account, 147 market.shortToken, 148 market.longToken, 149 shortTokenPrice, 150 longTokenPrice, 151 withdrawal.marketTokensLongAmount, 152 withdrawal.hasCollateralInETH, 153 cache.marketTokensLongUsd, 154 usdAdjustment * cache.marketTokensLongUsd.toInt256() / (cache. marketTokensLongUsd + cache.marketTokensShortUsd).toInt256() 155); 165 _ExecuteWithdrawalParams memory _params = _ExecuteWithdrawalParams(166 market, 167 withdrawal.account, 168 market.longToken, 169 market.shortToken, 170 longTokenPrice, 171 shortTokenPrice, 172 withdrawal.marketTokensShortAmount, 173 withdrawal.hasCollateralInETH, 174 cache.marketTokensShortUsd, 175 usdAdjustment * cache.marketTokensShortUsd.toInt256() / (cache. marketTokensLongUsd + cache.marketTokensShortUsd).toInt256() 186 GasUtils.payExecutionFee(187 params.dataStore, 188 params.withdrawalStore, 189 withdrawal.executionFee, 190 params.startingGas, 191 params.keeper, 192 withdrawal.account	183	Minor	Suboptimal		Use named args to avoid argument misplacing.
WithdrawalUtils.sol	97	97 bytes32 key = keccak256(abi.encodePacked(nonce));	184	Minor	Unclear behavior		It's not clear why to use hash of uint256 as a keym consider the usage of just uint256 nonce as a key to increase human readability of key and avoid redundant hashing.
WithdrawalUtils.sol	154, 175	154 usdAdjustment * cache.marketTokensLongUsd.toInt256() / (cache. marketTokensLongUsd + cache.marketTokensShortUsd).toInt256() 175 usdAdjustment * cache.marketTokensShortUsd.toInt256() / (cache. marketTokensLongUsd + cache.marketTokensShortUsd).toInt256()	185	Minor	Overflow/Underflow	Phantom overflow is possible here.	Consider using the “muldiv” function.
WithdrawalUtils.sol	231	231 PricingUtils.transferFees(	186	Minor	Suboptimal	Transferring fees on each transaction is inefficient.	Consider accumulating fees in the pool and allowing the fee receiver to kate them later.

EnumerableValues.sol	17-19, 30-32, 43-45	17 for (uint256 i = start; i < end; i++) {
		18 items[i - start] = set.at(i);
		19 }
		30 for (uint256 i = start; i < end; i++) {
		31 items[i - start] = set.at(i);
		32 }
		43 for (uint256 i = start; i < end; i++) {
		44 items[i - start] = set.at(i);
		45 }
			187	Minor	Suboptimal		Use unchecked declaration here to save gas because you know for sure that always i < type(uint256).max and i >= start

File	Lines	Code	CVF	Severity	Category	Description	Recommendation
SwapPricingUtils.sol	19–20	`19 address tokenA;` `20 address tokenB;`	188	Minor	Bad datatype		The type of these fields could be more specific.
SwapPricingUtils.sol	23–24, 28–31	`23 int256 usdDeltaForTokenA;` `24 int256 usdDeltaForTokenB;` `28 uint256 poolUsdForTokenA;` `29 uint256 poolUsdForTokenB;` `30 uint256 nextPoolUsdForTokenA;` `31 uint256 nextPoolUsdForTokenB;`	189	Minor	Documentation	The semantics of these fields is unclear.	Consider documenting.
SwapPricingUtils.sol	60-94	`60 function getUsdAdjustment(DataStore dataStore, address market, PoolParams memory poolParams) internal view returns (int256) {` `61 uint256 initialDiffUsd = Calc.diff(poolParams.poolUsdForTokenA, poolParams.poolUsdForTokenB);` `62 uint256 nextDiffUsd = Calc.diff(poolParams.nextPoolUsdForTokenA, poolParams.nextPoolUsdForTokenB);` `63` `64 // check whether an improvement in balance comes from causing the balance to switch sides` `65 // for example, if there is $2000 of ETH and $1000 of USDC in the pool` `66 // adding $1999 USDC into the pool will reduce absolute balance from $1000 to $999 but it does not` `67 // help rebalance the pool much, the isSameSideRebalance value helps avoid gaming using this case` `68 bool isSameSideRebalance = poolParams.poolUsdForTokenA <= poolParams.poolUsdForTokenB == poolParams.nextPoolUsdForTokenA <= poolParams.nextPoolUsdForTokenB;` `69 uint256 impactExponentFactor = dataStore.getUint(Keys.swapImpactExponentFactorKey(market));` `70` `71 if (isSameSideRebalance) {` `72 bool hasPositiveImpact = nextDiffUsd < initialDiffUsd;` `73 uint256 impactFactor = dataStore.getUint(Keys.swapImpactFactorKey(market, hasPositiveImpact));` `74` `75 return PricingUtils.getUsdAdjustmentForSameSideRebalance(` `76 initialDiffUsd,` `77 nextDiffUsd,` `78 hasPositiveImpact,` `79 impactFactor,` `80 impactExponentFactor` `81 );` `82 } else {` `83 uint256 positiveImpactFactor = dataStore.getUint(Keys.swapImpactFactorKey(market, true));` `84 uint256 negativeImpactFactor = dataStore.getUint(Keys.swapImpactFactorKey(market, false));` `85` `86 return PricingUtils.getUsdAdjustmentForCrossoverRebalance(` `87 initialDiffUsd,` `88 nextDiffUsd,` `89 positiveImpactFactor,` `90 negativeImpactFactor,` `91 impactExponentFactor` `92 );` `93 }` `94 }`	190	Minor	Suboptimal		Unclear logic, consider adding more explanations.
PositionPricingUtils.sol	19–20	`19 address longToken;` `20 address shortToken;`	191	Minor	Bad datatype		The type of these fields could be more specific.
PositionPricingUtils.sol	115-128	`115 function transferPositionFees(` `116 FeeReceiver feeReceiver,` `117 MarketToken marketToken,` `118 Position.Props memory position,` `119 bytes32 feeType,` `120 PositionFees memory fees` `121 ) internal returns (PositionFees memory) {` `122 if (fees.feeReceiverAmount > 0) {` `123 marketToken.transferOut(position.collateralToken, fees.feeReceiverAmount, address(feeReceiver));` `124 feeReceiver.notifyFeeReceived(feeType, position.collateralToken, fees.feeReceiverAmount);` `125 }` `126` `127 return fees;` `128 }`	192	Minor	Suboptimal		This function is never used, consider removing.
PositionPricingUtils.sol	149–150	`149 fees.feesForPool = fees.spreadAmount + fees.positionFeeAmount + fees.borrowingFeeAmount - fees.feeReceiverAmount;` `150 fees.totalNetCostAmount = fees.fundingFeeAmount - (fees.positionFeeAmount + fees.spreadAmount + fees.borrowingFeeAmount).toInt256();`	193	Minor	Suboptimal	The expression “fees.spreadAmount + fees.positionFeeAmount + fees.borrowingFeeAmount” is calculated twice.	Consider calculating once and reusing.
OrderUtils.sol	35	`35 address initialCollateralToken;`	194	Minor	Bad datatype		The type of this field could be more specific.

File	Lines	Code	CVF	Severity	Category	Description	Recommendation
OrderUtils.sol	94	`94        MarketUtils.getMarkets(marketStore, params.swapPath);`	195	Minor	Unclear behavior		The return array is ignored, consider rewriting the function getMarkets to validateMarkets only.
OrderUtils.sol	113	`113        bytes32 key = keccak256(abi.encodePacked(nonce));`	196	Minor	Suboptimal	Order key is just a hash of the nonce.	Just using the nonce as a key would be more efficient.

OrderUtils.sol	178, 183	`178    function isPositionOrder(Order.OrderType orderType) internal pure returns (bool) {`
		`183    function isIncreaseOrder(Order.OrderType orderType) internal pure returns (bool) {`
			197	Minor	Suboptimal	These two function and identical.	Consider merging them into one.

OrderUtils.sol	207, 213, 227, 241	`207        if (orderType == Order.OrderType.MarketIncrease || orderType == Order.OrderType.`
		`MarketDecrease) {`
		`213        if (orderType == Order.OrderType.LimitIncrease) {`
		`227        if (orderType == Order.OrderType.LimitDecrease) {`
		`241        if (orderType == Order.OrderType.StopLossDecrease) {`
			198	Minor	Readability		Should be “else if” for readability.

OrderUtils.sol	213, 227	`213        if (orderType == Order.OrderType.LimitIncrease) {`
		`227        if (orderType == Order.OrderType.LimitDecrease) {`
			199	Minor	Suboptimal	The bodies of these conditional statements are very similar.	Consider merging them together to avoid code duplication.

OrderUtils.sol	218, 221, 232, 235	`218                oracle.setSecondaryPrice(indexToken, acceptablePrice);`
		`221                oracle.setSecondaryPrice(indexToken, acceptablePrice);`
		`232                oracle.setSecondaryPrice(indexToken, acceptablePrice);`
		`235                oracle.setSecondaryPrice(indexToken, acceptablePrice);`
			200	Minor	Procedural		This should be done after the conditional statement to avoid code duplication.

OrderUtils.sol	264	`264        revert("OrderUtils: unsupported order type");`	201	Minor	Readability		Should be “else revert” for readability.
OrderUtils.sol	279	`279        if (orderType == Order.OrderType.LimitSwap) {`	202	Minor	Readability		Should be “else if” for readability.
OrderUtils.sol	286	`286        revertUnsupportedOrderType();`	203	Minor	Readability		Should be in an “else” branch.
OrderUtils.sol	306	`306        if (`	204	Minor	Readability		Should be “else if” for readability.
OrderUtils.sol	318	`318        revertUnsupportedOrderType();`	205	Minor	Readability		Should be in an “else” branch.
OrderUtils.sol	321	`321    function revertUnsupportedOrderType() internal pure {`	206	Minor	Procedural	This function is too simple to be extracted.	Consider removing this function.
Timelock.sol	8	`8contract Timelock {`	207	Minor	Documentation	The role of this contract is unclear. It's functionality it not related to time nor to locking. Also, its functionality doesn't seem complete. The contract is not used by other contracts.	Consider removing or explaining its role in a documentation comment.
Timelock.sol	9	`9    address public admin;`	208	Minor	Procedural		This variable should be declared as immutable.
Timelock.sol	9	`9    address public admin;`	209	Minor	Suboptimal		The admin role cannot be transferred, consider adding transferOwnership function or inherit from Ownable openzeppelin contract.
Timelock.sol	12	`12    mapping (string => bool) public allowedSlowKeys;`	210	Minor	Procedural	This mapping is never used.	Consider removing it.

Timelock.sol	29, 38	`29        string[6] memory allowedKeys = [`
		`38        for (uint256 i = 0; i < allowedKeys.length; i++) {`
			211	Minor	Suboptimal		The array and the loop are redundant here, as all the keys are hardcoded anyway. Just do six assignments instead.

Timelock.sol	43	`43    function fastSetUints(DataStore dataStore, string[] memory prefixes, bytes[] memory data,`
		`uint256[] memory values) external onlyAdmin {`
			212	Minor	Suboptimal		It would be more efficient to pass a single array of structs with three fields, rather then three parallel arrays. This would also make the length check unnecessary.

Timelock.sol	47, 58	`47            require(allowedFastKeys[prefix], "Timelock: invalid key");`
		`58        require(allowedFastKeys[prefix], "Timelock: invalid key");`
			213	Minor	Suboptimal	String error messages are inefficient.	Consider using named errors instead.

Router.sol	10	`10// users will approve this router for token spenditures`	214	Minor	Documentation	The word is misspelled.	Cosnider replacing with "expenditures"
Router.sol	14	`14    constructor(RoleStore _roleStore) RoleModule(_roleStore) {}`	215	Minor	Procedural		It is a good practice to put a comment into an empty block to explain why the block is empty.

Router.sol	16	`16    function pluginTransfer(address token, address account, address receiver, uint256 amount)`
		`external onlyRouterPlugin {`
			216	Minor	Bad datatype		The type of the “token” argument should be “IERC20”.

RoleStore.sol	18, 23, 27, 39, 43	`18    function grantRole(address account, bytes32 key) external onlyGov {`
		`23    function revokeRole(address account, bytes32 key) external onlyGov {`
		`27    function hasRole(address account, bytes32 key) external view returns (bool) {`
		`39    function getRoleMemberCount(bytes32 key) external view returns (uint256) {`
		`43    function getRoleMembers(bytes32 key, uint256 start, uint256 end) external view returns`
		`(address[] memory) {`
			217	Minor	Bad naming	The argument name “key” is too generic.	Consider renaming to “role”.

RoleStore.sol	18, 23	`18    function grantRole(address account, bytes32 key) external onlyGov {`
		`23    function revokeRole(address account, bytes32 key) external onlyGov {`
			218	Minor	Unclear behavior	These functions should emit some events.	

RoleStore.sol	18-21, 23-25	`18    function grantRole(address account, bytes32 key) external onlyGov {`
		`19        roles.add(key);`
		`20        roleMembers[key].add(account);`
		`21    }`
		`23    function revokeRole(address account, bytes32 key) external onlyGov {`
		`24        roleMembers[key].remove(account);`
		`25    }`
			219	Minor	Suboptimal		Consider strictly requiring that Role was not granted before via the check require (roleMembers[key].add(account), "already granted")

LiquidationUtils.sol	18	`18        address collateralToken,`	221	Minor	Bad datatype		The type of this argument could be more specific.
LiquidationUtils.sol	35	`35        order.setAcceptableUsdAdjustment(-type(int256).max);`	222	Minor	Suboptimal		The value “type(int256).min” would be more reasonable here.

File	Lines	Code	CVF	Severity	Category	Description	Recommendation

LiquidationUtils.sol	41–47	`41        OrderUtils.setExactOrderPrice(`
		`42            params.oracle,`
		`43            params.market.indexToken,`
		`44            params.order.orderType(),`
		`45            params.order.acceptablePrice(),`
		`46            params.order.isLong()`
		`47        );`
			223	Minor	Suboptimal	This call is an overkill here, as it is able to handle arbitrary order.	Could be replaced with: oracle.setSecondaryPrice (indexToken, oracle.getPrimaryPrice (indexToken));

LiquidationUtils.sol	63	`63        DecreaseOrderUtils.processOrder(params, true);`	224	Minor	Documentation	What if order can only be processed partialy?	Consider documenting this case.
Order.sol	88	`88        address initialCollateralToken;`	225	Minor	Bad datatype		The type of this field could be more specific.
Order.sol	92	`92    struct Numbers {`	226	Minor	Bad naming	The name "Numbers" says nothing.	Consider renaming to something meaningful.

Order.sol	93–94, 96	`93        uint256 sizeDeltaUsd;`
		`94        uint256 initialCollateralDeltaAmount;`
		`96        int256 acceptableUsdAdjustment;`
			227	Minor	Documentation	The semantics of these fields is unclear.	Consider documenting.

Order.sol	95, 97	`95        uint256 acceptablePrice;`
		`97        uint256 executionFee;`
			228	Minor	Documentation	The number format of these fields is unclear.	Consider documenting.

Order.sol	111	`111    struct Props {`	229	Minor	Bad naming	The name is too generic.	Consider renaming to “OrderProps” or just “Order”.
Order.sol	126	`126    function initialCollateralToken(Props memory props) internal pure returns (address) {`	230	Minor	Bad datatype		The return type could be more specific.
Order.sol	182	`182    function setInitialCollateralToken(Props memory props, address _value) internal pure {`	231	Minor	Bad datatype		The type of the “_value” argument could be more specific.

SwapOrderUtils.sol	17–18	`17        address firstMarket = params.order.swapPath()[0];`
		`18        address lastMarket = params.order.swapPath()[params.order.swapPath().length - 1];`
			232	Minor	Suboptimal	The expression “params.order.swapPath()” is calculated several times.	Consider calculating once and reusing.

SwapOrderUtils.sol	33-42	`33        (address tokenOut, uint256 outputAmount) = SwapUtils.swap(SwapUtils.SwapParams(`
		`34            params.dataStore,`
		`35            params.oracle,`
		`36            params.feeReceiver,`
		`37            params.order.initialCollateralToken(),`
		`38            params.order.initialCollateralDeltaAmount(),`
		`39            params.swapPathMarkets,`
		`40            params.order.minOutputAmount(),`
		`41            address(0)`
		`42        ));`
			233	Minor	Readability		Use named args to avoid misplacing.

OracleModule.sol	11	`11    // since re-entrancy could allow functions to be called with prices`	234	Minor	Documentation	This is not realy clear why is it necessary to update multiple storage slots rather than holding one reentry protection flag and passing desired values in memory values.	Consider explaining more.

IPriceFeed.sol	7, 11	`7        uint80 roundId,`
		`11        uint80 answeredInRound`
			235	Minor	Documentation	The difference between these two values is unclear.	Consider documenting.

IPriceFeed.sol	8	`8        int256 answer,`	236	Minor	Bad naming	The meaning of the word "answer" is not clear. Answer to which question is it?	Consider renaming to just "price".

IPriceFeed.sol	9–10	`9        uint256 startedAt,`
		`10        uint256 updatedAt,`
			237	Minor	Documentation	The semantics of these values is unclear.	Consider documenting.

MarketFactory.sol	13	`13    MarketStore public marketStore;`	238	Minor	Procedural		Consider declaring immutable.

MarketFactory.sol	20–22	`20        address indexToken,`
		`21        address longToken,`
		`22        address shortToken`
			239	Minor	Bad datatype		The type of these arguments could be more specific.

MarketFactory.sol	25	`25            "GMX_MARKET",`	240	Minor	Suboptimal		Consider declaring as a contract level constant.
MarketFactory.sol	47	`47    function addSwapMarket() external {}`	241	Minor	Procedural	This function does nothing.	Consider removing it.
MarketFactory.sol	47	`47    function addSwapMarket() external {}`	242	Minor	Procedural		It is a good practice to put a comment into an empty block to explain why the block is empty.
Market.sol	6	`6    struct Props {`	243	Minor	Bad naming	The name is too generic.	Consider renaming to “MarketProps” or just “Market”.
Market.sol	7	`7        address marketToken;`	244	Minor	Bad datatype		The type of this field should be “MarketToken”.

Market.sol	8–10	`8        address indexToken;`
		`9        address longToken;`
		`10        address shortToken;`
			245	Minor	Bad datatype		The type of these fields could be more specific.

Market.sol	8	`8        address indexToken;`	246	Minor	Unclear behavior	It's not clear should index token be one of "longToken" or "shortToken", so many tokens in the structure requires some explanations in comments.	
Market.sol	11	`11        bytes32[] data;`	247	Minor	Documentation	It's nor clear what potentially could be here.	Consider documenting.
Governable.sol	5	`5contract Governable {`	248	Minor	Suboptimal		This contract should be declared as “abstract”, as it is not supposed to be deployed as is, but rather inherited by other contracts.

File	Lines	Code	CVF	Severity	Category	Description	Recommendation
Governable.sol	6	`6 address public gov;`	249	Minor	Bad naming	Abbreviated public identifiers is a bad practice.	Consider using full words.
Governable.sol	8	`8 event SetGov(address prevGov, address nextGov);`	250	Minor	Bad naming		Events are usually named via nouns, such as "Governor".
Governable.sol	8	`8 event SetGov(address prevGov, address nextGov);`	251	Minor	Suboptimal	The "prevGov" parameter is redundant as its value could be derived from the previous events.	
Governable.sol	10	`10 error Unauthorized(address msgSender, string role);`	252	Minor	Suboptimal	The "role" parameter is redundant as its value is always the same.	It wouldn't be redundant in case this error would be a part of a authorization framework, and this contract would be among other applications of this framework.
Governable.sol	29	`29 gov = _gov;`	253	Minor	Suboptimal		Consider adding the requirement for prevGov != _gov to be sure that the value has changed.
Governable.sol	31	`31 emit SetGov(prevGov, _gov);`	254	Minor	Unclear behavior	This event is emitted even if nothing actually changed.	

Reader.sol	44–46	`44 uint256 longTokenPrice,`
		`45 uint256 shortTokenPrice,`
		`46 uint256 indexTokenPrice`
			255	Minor	Documentation	The number format of these arguments is unclear.	Consider documenting.

Reader.sol	47	`47 ) external view returns (uint256) {`	256	Minor	Documentation	The number format of the returned value is unclear.	Consider documenting.

DecreasePositionUtils.sol	155, 199	`155 ProcessCollateralValues memory values;`
		`199 PositionPricingUtils.PositionFees memory fees = processPositionCosts(params, prices, position, remainingCollateralAmount);`
			257	Minor	Suboptimal	These variables are redundant.	Just give names to the returned values and use them instead.

DecreasePositionUtils.sol	162	`162 remainingCollateralAmount -= collateralDeltaAmount;`	258	Minor	Suboptimal	Here an argument is used as a local variable. This is a bad practice that makes code harder to read.	Consider using a separate local variable instead.
DecreasePositionUtils.sol	189	`189 }`	259	Minor	Unclear behavior	The code below looks like it is always executed, while it is only executed when the position it not liquidated underwater.	Consider putting the rest of the function into an explicit "else" branch to make code easier to read.
DecreasePositionUtils.sol	268	`268 return emptyFees;`	260	Minor	Readability	Return statements in the middle of a function make code harder to read.	Consider refactoring.
DecreasePositionUtils.sol	272	`272 PricingUtils.transferFees(`	261	Minor	Suboptimal	Transferring fees on every position change is suboptimal.	Consider accumulating fees in the pool and allowing the fee receiver to take them later.
Precision.sol	10	`10 uint256 public constant FLOAT_PRECISION = 10 ** 30;`	262	Minor	Readability		This value could be rendered as '1e30".
Precision.sol	11	`11 uint256 public constant WEI_PRECISION = 10 ** 18;`	263	Minor	Readability		This value could be rendered as "1e18".
Precision.sol	10	`10 uint256 public constant FLOAT_PRECISION = 10 ** 30;`	264	Minor	Readability	This denominator is n on-standard. Standard denominators are 1e18 and 1e27.	Consider using a standard denominator.
Precision.sol	13	`13 uint256 public constant FLOAT_TO_WEI_DIVISOR = 10 ** 12;`	265	Minor	Readability		This value could be rendered as "1e12".
Precision.sol	13	`13 uint256 public constant FLOAT_TO_WEI_DIVISOR = 10 ** 12;`	266	Minor	Suboptimal		The value for this constant could be calculated as: FLOAT_PRECISION / WEI_PRECISION
Precision.sol	10	`10 uint256 public constant FLOAT_PRECISION = 10 ** 30;`	267	Minor	Bad naming	The constant name is misleading, as this is actually a fixed-point precision, rather than floating point.	
Precision.sol	20	`20 return amount.toInt256() * factor / FLOAT_PRECISION.toInt256();`	268	Minor	Suboptimal	Applying the "toInt256" function to a constant is waste of gas.	Consider using plain conversion.

Precision.sol	20	`20 return amount.toInt256() * factor / FLOAT_PRECISION.toInt256();`	269	Minor	Overflow/Underflow	Converting the amount to "int256" before applying factor makes phantom overflow more likely.	Consider doing like this:
							if (factor >= 0)
							  return applyFactor (amount, uint256 (factor)).toInt256 ();
							else
							  return -applyFactor (amount, uint256 (-factor)).toInt256 ();

PricingUtils.sol	39, 41	`39 uint256 deltaDiffUsd = Calc.diff(positiveImpactUsd, negativeImpactUsd);`
		`41 int256 usdAdjustment = Calc.toSigned(deltaDiffUsd, positiveImpactUsd > negativeImpactUsd);`
			270	Minor	Suboptimal	This pair of operations is equivalent to a plain signed subtraction.	

PricingUtils.sol	51	``51 // `PRBMathUD60x18.pow` doesn't work for `x` less than one``	271	Minor	Suboptimal		Consider using a signed version of PRB Math to bypass this limitation.
PricingUtils.sol	69	`69 address marketToken,`	272	Minor	Bad datatype		The type of this argument should be "MarketToken".
Calc.sol	17	`17 return a + b.abs();`	273	Minor	Suboptimal		The "abs" call is redundant here. Just do: a + uint256 (b)
Calc.sol	20	`20 return a - b.abs();`	274	Minor	Suboptimal	The "abs" call is inefficient here, as "b" is known to be negative.	Consider doing: return a - uint256 (-b);

Calc.sol	24, 29, 31	`24 return a + SafeCast.toInt256(b);`
		`29 return SafeCast.toInt256(a);`
		`31 return -SafeCast.toInt256(a);`
			275	Minor	Suboptimal		These lines could be simplified by specifying "using SafeCase for uint256;".

Withdrawal.sol	8	`8 struct Props {`	276	Minor	Bad naming	The name is too generic.	Consider renaming to "WithdrawalProps" or just "Withdrawal".

File	Lines	Code	CVF	Severity	Category	Description	Recommendation

Bits.sol	8, 10, 12	`8    uint256 constant public BITMASK_16 = ~uint256(0) >> (256 - 16);`
		`10    uint256 constant public BITMASK_32 = ~uint256(0) >> (256 - 32);`
		`12    uint256 constant public BITMASK_64 = ~uint256(0) >> (256 - 64);`
			277	Minor	Readability	These formulas don't make the code easier to read.	Consider just using "type(uint16)max', "type(uint320.max", and "type(uint64).max".

RoleModule.sol	23, 30, 35, 40, 45, 50	`23    modifier onlyController() {`
		`30    modifier onlyRouterPlugin() {`
		`35    modifier onlyMarketKeeper() {`
		`40    modifier onlyOrderKeeper() {`
		`45    modifier onlyPricingKeeper() {`
		`50    modifier onlyLiquidationKeeper() {`
			278	Minor	Readability		These modifiers could be replaced with a single modifier:
							onlyRole (bytes32 role);

RoleModule.sol	31, 36, 41, 46, 51	`31        require(roleStore.hasRole(msg.sender, Role.ROUTER_PLUGIN), "Role: ROUTER_PLUGIN");`
		`36        require(roleStore.hasRole(msg.sender, Role.MARKET_KEEPER), "Role: MARKET_KEEPER");`
		`41        require(roleStore.hasRole(msg.sender, Role.ORDER_KEEPER), "Role: ORDER_KEEPER");`
		`46        require(roleStore.hasRole(msg.sender, Role.PRICING_KEEPER), "Role: PRICING_KEEPER");`
		`51        require(roleStore.hasRole(msg.sender, Role.LIQUIDATION_KEEPER), "Role:`
		`LIQUIDATION_KEEPER");`
			279	Minor	Suboptimal	String error messages are inefficient.	Consider using named error instead.

Position.sol	6	`6    struct Props {`	280	Minor	Bad naming	The name is too generic.	Consider renaming to "PositionProps" or just "Position".
Position.sol	9	`9        address collateralToken;`	281	Minor	Bad datatype		The type of this field could be more specific.
NonceUtils.sol	13	`13    function incrementNonce(DataStore dataStore) internal returns (uint256) {`	282	Minor	Documentation	The semantics of the returned value is unclear.	Consider documenting.

MarketToken.sol	9–10	`9    constructor(RoleStore _roleStore) ERC20("GMX Synthetic Market", "GD") Bank(_roleStore) {`
		`10    }`
			283	Minor	Procedural		It is a good practice to put a comment into an empty block to explain why the block is empty.

FeeReceiver.sol	Tue May 10 2022 00:00:00 GMT-0700 (Pacific Daylight Time)		284	Minor	Suboptimal	It's not efficient to use separate contract with an external call just to emit event.	Consider emiting the event inside every contract where it is needed.
FeeReceiver.sol	6	`6    event FeeReceived(bytes32 key, address token, uint256 amount);`	285	Minor	Bad naming		Events are usually named via nouns, such as "Fee" or "ReceivedFee".
FeeReceiver.sol	6	`6    event FeeReceived(bytes32 key, address token, uint256 amount);`	286	Minor	Procedural		The first two parameters should be indexed.
FeeReceiver.sol	6	`6    event FeeReceived(bytes32 key, address token, uint256 amount);`	287	Minor	Bad datatype		The type of the "token" parameter could be more specific.
FeeReceiver.sol	8	`8    function notifyFeeReceived(bytes32 key, address token, uint256 amount) external {`	288	Minor	Bad datatype		The type of the "token" argument could be more specific.

WithdrawalHandler.sol	24–28	`24    DataStore public dataStore;`
		`25    WithdrawalStore public withdrawalStore;`
		`26    MarketStore public marketStore;`
		`27    Oracle public oracle;`
		`28    FeeReceiver public feeReceiver;`
			289	Minor	Procedural		These variables should be declared as immutable.

WithdrawalHandler.sol	61-74, 127-137	`61        WithdrawalUtils.CreateWithdrawalParams memory params = WithdrawalUtils.`
		`CreateWithdrawalParams(`
		`62            dataStore,`
		`63            withdrawalStore,`
		`64            marketStore,`
		`65            account,`
		`66            market,`
		`67            marketTokensLongAmount,`
		`68            marketTokensShortAmount,`
		`69            minLongTokenAmount,`
		`70            minShortTokenAmount,`
		`71            hasCollateralInETH,`
		`72            executionFee,`
		`73            EthUtils.weth(dataStore)`
		`74        );`
		`127        WithdrawalUtils.ExecuteWithdrawalParams memory params = WithdrawalUtils.`
		`ExecuteWithdrawalParams(`
		`128            dataStore,`
		`129            withdrawalStore,`
		`130            marketStore,`
		`131            oracle,`
		`132            feeReceiver,`
		`133            key,`
		`134            oracleBlockNumbers,`
		`135            keeper,`
		`136            startingGas`
		`137        );`
			290	Minor	Readability		Consider using named args.

WithdrawalHandler.sol	96	`96            if (keccak256(abi.encodePacked(reason)) == Keys.ORACLE_ERROR_KEY) {`	291	Minor	Suboptimal		Consider direct comparison with a ORACLE_ERROR.

OrderHandler.sol	28–33	`28    DataStore public dataStore;`
		`29    MarketStore public marketStore;`
		`30    OrderStore public orderStore;`
		`31    PositionStore public positionStore;`
		`32    Oracle public oracle;`
		`33    FeeReceiver public feeReceiver;`
			292	Minor	Procedural		These variables should be declared as immutable.

File	Lines	Code	CVF	Severity	Category	Description	Recommendation

OrderHandler.sol	56, 71, 105, 130	`56 function createOrder(`
		`71 function executeOrder(`
		`105 function updateOrder(`
		`130 function cancelOrder(bytes32 key) external {`
			293	Minor	Unclear behavior	These functions should emit some events.	

OrderHandler.sol	196, 201	`196 if (OrderUtils.isDecreaseOrder(params.order.orderType())) {`
		`201 if (OrderUtils.isSwapOrder(params.order.orderType())) {`
			294	Minor	Readability		Should be "else if" for readability.

OrderHandler.sol	206	`206 OrderUtils.revertUnsupportedOrderType();`	295	Minor	Readability		This line should be in an "else" branch for readability.
OrderHandler.sol	238	`238 OrderUtils.validateNonEmptyOrder(params.order);`	296	Minor	Procedural		This check should be performed earlier.
StrictBank.sol	15	`15 mapping (address => uint256) public tokenBalances;`	297	Minor	Bad datatype		The key type should be "IERC20".
StrictBank.sol	17	`17 constructor(RoleStore _roleStore) Bank(_roleStore) {}`	298	Minor	Procedural		It is a good practice to put a comment into an empty block to explain why the block is empty.

StrictBank.sol	19, 23, 31	`19 function recordTransferIn(address token) external onlyController returns (uint256) {`
		`23 function _recordTransferIn(address token) internal returns (uint256) {`
		`31 function _afterTransferOut(address token) internal override {`
			299	Minor	Bad datatype		The type of the "token" argument should be "IERC20".

StrictBank.sol	23-29	`23 function _recordTransferIn(address token) internal returns (uint256) {`
		`24 uint256 prevBalance = tokenBalances[token];`
		`25 uint256 nextBalance = IERC20(token).balanceOf(address(this));`
		`26 tokenBalances[token] = nextBalance;`
		`27`
		`28 return nextBalance - prevBalance;`
		`29 }`
			300	Minor	Documentation	WARNING This mechanic will not work in case of reflactions deflationary tokens.	Consider documenting this limitation.

StrictBank.sol	32	`32 tokenBalances[token] = IERC20(token).balanceOf(address(this));`	301	Minor	Suboptimal		It would be more efficient to just subtract the known outgoing transfer amount from the stored balance. Calling an external contract is expensive.

DepositHandler.sol	22–26	`22 DataStore public dataStore;`
		`23 DepositStore public depositStore;`
		`24 MarketStore public marketStore;`
		`25 Oracle public oracle;`
		`26 FeeReceiver public feeReceiver;`
			302	Minor	Procedural		These variables should be declared as immutable.

DepositHandler.sol	44	`44 require(msg.sender == EthUtils.weth(dataStore), "DepositHandler: invalid sender");`	303	Minor	Suboptimal	String error messages are suboptimal.	Consider using named errors instead.
DepositHandler.sol	53	`53 ) external nonReentrant onlyController returns (bytes32) {`	304	Minor	Documentation	The semantics of the returned value is unclear.	Consider giving a descriptive names to the returned value and/or adding a documentation comment.
DepositHandler.sol	54	`54 FeatureUtils.validateFeature(dataStore, Keys.createDepositFeatureKey(address(this)));`	305	Minor	Procedural		The feature key could be precomputed in the constructor and stored in an immutable variable.

DepositHandler.sol	56-65	`56 DepositUtils.CreateDepositParams memory params = DepositUtils.CreateDepositParams(`
		`57 dataStore,`
		`58 depositStore,`
		`59 marketStore,`
		`60 account,`
		`61 market,`
		`62 minMarketTokens,`
		`63 hasCollateralInETH,`
		`64 executionFee,`
		`65 EthUtils.weth(dataStore)`
			306	Minor	Readability		Consider using named arguments to avoid mistakes.

DepositHandler.sol	85-87	`85 if (keccak256(abi.encodePacked(reason)) == Keys.ORACLE_ERROR_KEY) {`
		`86 revert(reason);`
		`87 }`
			307	Minor	Documentation	How to cancelDeposit in case if oracle system is broken and this error is permanent?	Consider documenting or implementing canceling mechanism.

DepositHandler.sol	85	`85 if (keccak256(abi.encodePacked(reason)) == Keys.ORACLE_ERROR_KEY) {`	308	Minor	Suboptimal		Consider comparing directly reason with Keys.ORACLE_ERROR without hashing.
DepositHandler.sol	107	`107 onlySelf`	309	Minor	Suboptimal		This check is not needed if you replace "public" modifier with "internal".
EthUtils.sol	9	`9 function weth(DataStore dataStore) internal view returns (address) {`	310	Minor	Bad datatype		The return type should be "IWETH".
EthUtils.sol	10	`10 return dataStore.getAddress(Keys.WETH);`	311	Minor	Suboptimal		The safer approach would be to use immutable WETH address rather than dynamically set value inside dataStore.
DepositUtils.sol	33	`33 address weth;`	312	Minor	Bad datatype		The type of this field should be "IWETH".
DepositUtils.sol	48	`48 struct _ExecuteDepositParams {`	313	Minor	Bad naming		The name is too similar to "ExecuteDepositParams" consider renaming

DepositUtils.sol	51–52	`51 address tokenIn;`
		`52 address tokenOut;`
			314	Minor	Bad datatype		The type of these fields could be more specific.

DepositUtils.sol	53–54	`53 uint256 tokenInPrice;`
		`54 uint256 tokenOutPrice;`
			315	Minor	Documentation	The number format of these fields is unclear.	Consider documenting.

DepositUtils.sol	56	`56 int256 usdAdjustment;`	316	Minor	Documentation		The meaning of this attribute is not clear, consider documenting.
DepositUtils.sol	61	`61 function createDeposit(CreateDepositParams memory params) external returns (bytes32) {`	317	Minor	Documentation	The semantics of the returned value is unclear.	Consider giving a descriptive name to the returned value and/or adding a documentation comment.

File	Lines	Code	CVF	Severity	Category	Description	Recommendation
DepositUtils.sol	61	`61    function createDeposit(CreateDepositParams memory params) external returns (bytes32) {`	318	Minor	Unclear behavior	This function should emit some event.	

DepositUtils.sol	68-75	`68        if (market.longToken == params.weth) {`
		`69            longTokenAmount -= params.executionFee;`
		`70        } else if (market.shortToken == params.weth) {`
		`71            shortTokenAmount -= params.executionFee;`
		`72        } else {`
		`73            uint256 wethAmount = params.depositStore.recordTransferIn(params.weth);`
		`74            require(wethAmount == params.executionFee, "DepositUtils: invalid wethAmount");`
		`75        }`
			319	Minor	Suboptimal		It's safer to use storage variable weth rather than user input argument which may potentially be manipulated.

DepositUtils.sol	68-75	`68        if (market.longToken == params.weth) {`
		`69            longTokenAmount -= params.executionFee;`
		`70        } else if (market.shortToken == params.weth) {`
		`71            shortTokenAmount -= params.executionFee;`
		`72        } else {`
		`73            uint256 wethAmount = params.depositStore.recordTransferIn(params.weth);`
		`74            require(wethAmount == params.executionFee, "DepositUtils: invalid wethAmount");`
		`75        }`
			320	Minor	Documentation		Consider adding a comment that the validity of user-passed executionFee is checked below.

DepositUtils.sol	69, 71, 73-74	`69            longTokenAmount -= params.executionFee;`
		`71            shortTokenAmount -= params.executionFee;`
		`73            uint256 wethAmount = params.depositStore.recordTransferIn(params.weth);`
		`74            require(wethAmount == params.executionFee, "DepositUtils: invalid wethAmount");`
			321	Minor	Suboptimal		Consider emitting an event ExecutionFeeCharged to be able to track it in history.

DepositUtils.sol	77-87	`77        Deposit.Props memory deposit = Deposit.Props(`
		`78            params.account,`
		`79            market.marketToken,`
		`80            longTokenAmount,`
		`81            shortTokenAmount,`
		`82            params.minMarketTokens,`
		`83            block.number,`
		`84            params.hasCollateralInETH,`
		`85            params.executionFee,`
		`86            new bytes32[](0)`
		`87        );`
			322	Minor	Readability		Consider the usage of named arguments to avoid misplacing values.

DepositUtils.sol	93	`93        bytes32 key = keccak256(abi.encodePacked(nonce));`	323	Minor	Unclear behavior	It is not clear why calculating bytes32 hash over uint256 value is needed. It makes human readability worse.	Consider the usage of uint256 nonce as a key.
DepositUtils.sol	100	`100    function executeDeposit(ExecuteDepositParams memory params) internal {`	324	Minor	Unclear behavior	The function should emit an event.	
DepositUtils.sol	105	`105            revert(Keys.ORACLE_ERROR);`	325	Minor	Documentation		Consider the usage of more descriptive error name.
DepositUtils.sol	108	`108        Market.Props memory market = params.marketStore.get(deposit.market);`	326	Minor	Suboptimal		Market here potentially coould be removed, consider adding the check of the market validity.

File	Lines	Code	CVF	Severity	Category	Description	Recommendation
DepositUtils.sol	119-128, 141-150, 158-167, 178-185, 201-207, 211-217, 222-229	119 SwapPricingUtils.GetSwapPricingParams(120 params.dataStore, 121 market.marketToken, 122 market.longToken, 123 market.shortToken, 124 longTokenPrice, 125 shortTokenPrice, 126 (deposit.longTokenAmount * longTokenPrice).toInt256(), 127 (deposit.shortTokenAmount * shortTokenPrice).toInt256() 128) 141 _ExecuteDepositParams memory _params = _ExecuteDepositParams(142 market, 143 deposit.account, 144 market.longToken, 145 market.shortToken, 146 longTokenPrice, 147 shortTokenPrice, 148 deposit.longTokenAmount, 149 usdAdjustment * longTokenUsd.toInt256() / (longTokenUsd + shortTokenUsd). toInt256() 150); 158 _ExecuteDepositParams memory _params = _ExecuteDepositParams(159 market, 160 deposit.account, 161 market.shortToken, 162 market.longToken, 163 shortTokenPrice, 164 longTokenPrice, 165 deposit.shortTokenAmount, 166 usdAdjustment * shortTokenUsd.toInt256() / (longTokenUsd + shortTokenUsd). toInt256() 167); 178 GasUtils.payExecutionFee(179 params.dataStore, 180 params.depositStore, 181 deposit.executionFee, 182 params.startingGas, 183 params.keeper, 184 deposit.account 185); 201 depositStore.transferOut(202 EthUtils.weth(dataStore), 203 market.longToken, 204 deposit.longTokenAmount, 205 deposit.account, 206 deposit.hasCollateralInETH 207); 211 depositStore.transferOut(212 EthUtils.weth(dataStore), 213 market.shortToken, 214 deposit.shortTokenAmount, 215 deposit.account, 216 deposit.hasCollateralInETH 217); 222 GasUtils.payExecutionFee(223 dataStore, 224 depositStore, 225 deposit.executionFee, 226 startingGas, 227 keeper, 228 deposit.account 229);	327	Minor	Readability		Consider using named arguments.
DepositUtils.sol	126–127	126 (deposit.longTokenAmount * longTokenPrice).toInt256(), 127 (deposit.shortTokenAmount * shortTokenPrice).toInt256()	328	Minor	Procedural	The products here were already calculated above.	Consider using the already calculated values.
DepositUtils.sol	137	137 // this will not work correctly for tokens with a burn mechanism, those need to be separately handled	329	Minor	Documentation		The phrase "need to be separately" handled is not clear.
DepositUtils.sol	188	188 function cancelDeposit(	330	Minor	Unclear behavior	The function should emit an event	
DepositUtils.sol	199	199 Market.Props memory market = marketStore.get(deposit.market);	331	Minor	Suboptimal		Consider checking the validity of market.

File	Lines	Code	CVF	Severity	Category	Description	Recommendation
DepositUtils.sol	200-217	`200 if (deposit.longTokenAmount > 0) { 201 depositStore.transferOut(202 EthUtils.weth(dataStore), 203 market.longToken, 204 deposit.longTokenAmount, 205 deposit.account, 206 deposit.hasCollateralInETH 207); 208 } 209 210 if (deposit.shortTokenAmount > 0) { 211 depositStore.transferOut(212 EthUtils.weth(dataStore), 213 market.shortToken, 214 deposit.shortTokenAmount, 215 deposit.account, 216 deposit.hasCollateralInETH 217);`	332	Minor	Documentation	ExecutionFee will not be returned.	Consider documenting this behaviour in comments or adding the returingn of ExecutionFee.
DepositUtils.sol	289	`289 mintAmount += MarketUtils.usdToMarketTokenAmount(`	333	Minor	Suboptimal	The “+=” operator here is confusing, as the “mintAmount” value is guaranteed to be zero here.	Consider using “=” instead.
DepositStore.sol	15	`15 constructor(RoleStore _roleStore) StrictBank(_roleStore) {}`	334	Minor	Procedural		It is a good practice to put a comment into an empty block to explain why the block is empty.
DepositStore.sol	17	`17 function set(bytes32 key, Deposit.Props memory deposit) external onlyController {`	335	Minor	Bad naming		The name “set” is associated with setting a flag or an atomic variable, while here a value for a key is set. Such functions are usually named “put”. Also, consider name “store”.
DepositStore.sol	19, 24	`19 depositKeys.add(key); 24 depositKeys.remove(key);`	336	Minor	Suboptimal		Consider emitting an event.

File	Lines	Code	CVF	Severity	Category	Description	Recommendation

Keys.sol	6–7, 9–10, 12–13, 15–18, 20, 23, 25, 27, 29, 31, 33, 35, 37–38, 40–41, 43–49, 51–52, 54–78, 80–81, 83–84, 86–87, 89–90	6 bytes32 public constant WETH = keccak256("WETH");
		7 bytes32 public constant NONCE = keccak256("NONCE");
		9 bytes32 public constant CREATE_DEPOSIT_FEATURE = keccak256("CREATE_DEPOSIT_FEATURE");
		10 bytes32 public constant EXECUTE_DEPOSIT_FEATURE = keccak256("EXECUTE_DEPOSIT_FEATURE");
		12 bytes32 public constant CREATE_WITHDRAWAL_FEATURE = keccak256("CREATE_WITHDRAWAL_FEATURE");
		13 bytes32 public constant EXECUTE_WITHDRAWAL_FEATURE = keccak256
		("EXECUTE_WITHDRAWAL_FEATURE");
		15 bytes32 public constant CREATE_ORDER_FEATURE = keccak256("CREATE_ORDER_FEATURE");
		16 bytes32 public constant EXECUTE_ORDER_FEATURE = keccak256("EXECUTE_ORDER_FEATURE");
		17 bytes32 public constant UPDATE_ORDER_FEATURE = keccak256("UPDATE_ORDER_FEATURE");
		18 bytes32 public constant CANCEL_ORDER_FEATURE = keccak256("CANCEL_ORDER_FEATURE");
		20 bytes32 public constant LIQUIDATE_POSITION_FEATURE = keccak256
		("LIQUIDATE_POSITION_FEATURE");
		23 bytes32 public constant MIN_ORACLE_SIGNERS = keccak256("MIN_ORACLE_SIGNERS");
		25 bytes32 public constant MIN_ORACLE_BLOCK_CONFIRMATIONS = keccak256
		("MIN_ORACLE_BLOCK_CONFIRMATIONS");
		27 bytes32 public constant MAX_ORACLE_BLOCK_AGE = keccak256("MAX_ORACLE_BLOCK_AGE");
		29 bytes32 public constant FEE_RECEIVER_DEPOSIT_FACTOR = keccak256
		("FEE_RECEIVER_DEPOSIT_FACTOR");
		31 bytes32 public constant FEE_RECEIVER_WITHDRAWAL_FACTOR = keccak256
		("FEE_RECEIVER_WITHDRAWAL_FACTOR");
		33 bytes32 public constant FEE_RECEIVER_SWAP_FACTOR = keccak256("FEE_RECEIVER_SWAP_FACTOR");
		35 bytes32 public constant FEE_RECEIVER_POSITION_FACTOR = keccak256
		("FEE_RECEIVER_POSITION_FACTOR");
		37 bytes32 public constant ESTIMATED_FEE_BASE_GAS_LIMIT = keccak256
		("ESTIMATED_FEE_BASE_GAS_LIMIT");
		38 bytes32 public constant ESTIMATED_FEE_MULTIPLIER_FACTOR = keccak256
		("ESTIMATED_FEE_MULTIPLIER_FACTOR");
		40 bytes32 public constant EXECUTION_FEE_BASE_GAS_LIMIT = keccak256
		("EXECUTION_FEE_BASE_GAS_LIMIT");
		41 bytes32 public constant EXECUTION_FEE_MULTIPLIER_FACTOR = keccak256
		("EXECUTION_FEE_MULTIPLIER_FACTOR");
		43 bytes32 public constant DEPOSIT_GAS_LIMIT = keccak256("DEPOSIT_GAS_LIMIT");
		44 bytes32 public constant WITHDRAWAL_GAS_LIMIT = keccak256("WITHDRAWAL_GAS_LIMIT");
		45 bytes32 public constant SINGLE_SWAP_GAS_LIMIT = keccak256("SINGLE_SWAP_GAS_LIMIT");
		46 bytes32 public constant INCREASE_ORDER_GAS_LIMIT = keccak256("INCREASE_ORDER_GAS_LIMIT");
		47 bytes32 public constant DECREASE_ORDER_GAS_LIMIT = keccak256("DECREASE_ORDER_GAS_LIMIT");
		48 bytes32 public constant SWAP_ORDER_GAS_LIMIT = keccak256("SWAP_ORDER_GAS_LIMIT");
		49 bytes32 public constant CANCELLATION_GAS_LIMIT = keccak256("CANCELLATION_GAS_LIMIT");
		51 bytes32 public constant MAX_LEVERAGE = keccak256("MAX_LEVERAGE");
		52 bytes32 public constant MIN_COLLATERAL_USD = keccak256("MIN_COLLATERAL_USD");
		54 string public constant POSITION_IMPACT_FACTOR = "POSITION_IMPACT_FACTOR";
		55 string public constant POSITION_IMPACT_EXPONENT_FACTOR = "POSITION_IMPACT_EXPONENT_FACTOR";
		56 string public constant POSITION_SPREAD_FACTOR = "POSITION_SPREAD_FACTOR";
		57 string public constant POSITION_FEE_FACTOR = "POSITION_FEE_FACTOR";
		58 string public constant SWAP_IMPACT_FACTOR = "SWAP_IMPACT_FACTOR";
		59 string public constant SWAP_IMPACT_EXPONENT_FACTOR = "SWAP_IMPACT_EXPONENT_FACTOR";
		60 string public constant SWAP_SPREAD_FACTOR = "SWAP_SPREAD_FACTOR";
		61 string public constant SWAP_FEE_FACTOR = "SWAP_FEE_FACTOR";
		62 string public constant ORACLE_PRECISION = "ORACLE_PRECISION";
		63 string public constant OPEN_INTEREST = "OPEN_INTEREST";
		64 string public constant OPEN_INTEREST_IN_TOKENS = "OPEN_INTEREST_IN_TOKENS";
		65 string public constant COLLATERAL_SUM = "COLLATERAL_SUM";
		66 string public constant POOL_AMOUNT = "POOL_AMOUNT";
		67 string public constant SWAP_IMPACT_POOL_AMOUNT = "SWAP_IMPACT_POOL_AMOUNT";
		68 string public constant PRICE_FEED = "PRICE_FEED";
		69 string public constant PRICE_FEED_PRECISION = "PRICE_FEED_PRECISION";
		70 string public constant STABLE_PRICE = "STABLE_PRICE";
		71 string public constant RESERVE_FACTOR = "RESERVE_FACTOR";
		72 string public constant FUNDING_FACTOR = "FUNDING_FACTOR";
		73 string public constant CUMULATIVE_FUNDING_FACTOR = "CUMULATIVE_FUNDING_FACTOR";
		74 string public constant CUMULATIVE_FUNDING_FACTOR_UPDATED_AT =
		"CUMULATIVE_FUNDING_FACTOR_UPDATED_AT";
		75 string public constant BORROWING_FACTOR = "BORROWING_FACTOR";
		76 string public constant CUMULATIVE_BORROWING_FACTOR = "CUMULATIVE_BORROWING_FACTOR";
		77 string public constant CUMULATIVE_BORROWING_FACTOR_UPDATED_AT =
		"CUMULATIVE_BORROWING_FACTOR_UPDATED_AT";
		78 string public constant TOTAL_BORROWING = "TOTAL_BORROWING";
		80 string public constant ORACLE_ERROR = "ORACLE_ERROR";
		81 bytes32 public constant ORACLE_ERROR_KEY = keccak256(abi.encodePacked(ORACLE_ERROR));
		83 string public constant EMPTY_POSITION_ERROR = "EMPTY_POSITION_ERROR";
		84 bytes32 public constant EMPTY_POSITION_ERROR_KEY = keccak256(abi.encodePacked
		(EMPTY_POSITION_ERROR));
		86 string public constant UNACCEPTABLE_USD_ADJUSTMENT_ERROR =
		"UNACCEPTABLE_USD_ADJUSTMENT_ERROR";
		87 bytes32 public constant UNACCEPTABLE_USD_ADJUSTMENT_ERROR_KEY = keccak256(abi.encodePacked
		(UNACCEPTABLE_USD_ADJUSTMENT_ERROR));
		89 string public constant INSUFFICIENT_SWAP_OUTPUT_AMOUNT_ERROR =
		"INSUFFICIENT_SWAP_OUTPUT_AMOUNT_ERROR";
		90 bytes32 public constant INSUFFICIENT_SWAP_OUTPUT_AMOUNT_ERROR_KEY = keccak256(abi.
		encodePacked(INSUFFICIENT_SWAP_OUTPUT_AMOUNT_ERROR));
			337	Minor	Procedural	Public constants don't have much sense in a library.	Consider declaring as internal.

File	Lines	Code	CVF	Severity	Category	Description	Recommendation
Keys.sol	81	`81    bytes32 public constant ORACLE_ERROR_KEY = keccak256(abi.encodePacked(ORACLE_ERROR));`	338	Minor		Consider using the “ORACLE_ERROR” constant as the hash argument.	

Keys.sol	81, 84, 87, 90, 108, 114, 120, 126	`81    bytes32 public constant ORACLE_ERROR_KEY = keccak256(abi.encodePacked(ORACLE_ERROR));`
		`84    bytes32 public constant EMPTY_POSITION_ERROR_KEY = keccak256(abi.encodePacked (EMPTY_POSITION_ERROR));`
		`87    bytes32 public constant UNACCEPTABLE_USD_ADJUSTMENT_ERROR_KEY = keccak256(abi.encodePacked (UNACCEPTABLE_USD_ADJUSTMENT_ERROR));`
		`90    bytes32 public constant INSUFFICIENT_SWAP_OUTPUT_AMOUNT_ERROR_KEY = keccak256(abi. encodePacked(INSUFFICIENT_SWAP_OUTPUT_AMOUNT_ERROR));`
		`108        return keccak256(abi.encodePacked(`
		`114        return keccak256(abi.encodePacked(`
		`120        return keccak256(abi.encodePacked(`
		`126        return keccak256(abi.encodePacked(`
			339	Minor	Suboptimal		Here, “abi.encodePacked” calls could be replaced with simple conversions to “bytes”.

Keys.sol	84	`84    bytes32 public constant EMPTY_POSITION_ERROR_KEY = keccak256(abi.encodePacked (EMPTY_POSITION_ERROR));`	340	Minor	Suboptimal		Consider using the “EMPTY_POSITION_ERROR” constant as the hash argument.
Keys.sol	87	`87    bytes32 public constant UNACCEPTABLE_USD_ADJUSTMENT_ERROR_KEY = keccak256(abi.encodePacked (UNACCEPTABLE_USD_ADJUSTMENT_ERROR));`	341	Minor	Suboptimal		Consider using the “UNACCEPTABLE_USD_ADJUSTMENT_ERROR” constant as the hash argument.
Keys.sol	90	`90    bytes32 public constant INSUFFICIENT_SWAP_OUTPUT_AMOUNT_ERROR_KEY = keccak256(abi. encodePacked(INSUFFICIENT_SWAP_OUTPUT_AMOUNT_ERROR));`	342	Minor	Suboptimal		Consider using the “INSUFFICIENT_SWAP_OUTPUT_AMOUNT_ERROR” constant as the hash argument.

Keys.sol	302-304	`302    function impactPoolAmountKey(address market, address token) internal pure returns (bytes32) {`
		`303        return keccak256(abi.encodePacked(`
		`304            SWAP_IMPACT_POOL_AMOUNT,`
			343	Minor	Procedural		Naming inconvenience, consider usage of the same prefix Swap both for the constant name and for the function name.

LiquidationHandler.sol	25–29	`25    DataStore public dataStore;`
		`26    MarketStore public marketStore;`
		`27    PositionStore public positionStore;`
		`28    Oracle public oracle;`
		`29    FeeReceiver public feeReceiver;`
			344	Minor	Procedural		These variables should be declared as immutable.

LiquidationHandler.sol	49	`49        address collateralToken,`	345	Minor	Bad datatype		The type of this argument could be more specific.
FeatureUtils.sol	12	`12        return dataStore.getBool(key);`	346	Minor	Suboptimal	This basically allows checking arbitrary keys, not only feature-related.	Consider hashing the key here with some unique salt to guarantee that only features could be checked.

DataStore.sol	8–12	`8    mapping(bytes32 => uint256) public uintValues;`
		`9    mapping(bytes32 => int256) public intValues;`
		`10    mapping(bytes32 => address) public addressValues;`
		`11    mapping(bytes32 => bytes32) public dataValues;`
		`12    mapping(bytes32 => bool) public boolValues;`
			347	Minor	Documentation	The semantics of the keys in this mapping is unclear.	Consider documenting.

DataStore.sol	14	`14    constructor(RoleStore _roleStore) RoleModule(_roleStore) {}`	348	Minor	Procedural		It is a good practice to put a comment into an empty block to explain why the block is empty.

DataStore.sol	16, 37, 58, 67, 76	`16    function getUint(bytes32 key) external view returns (uint256) {`
		`37    function getInt(bytes32 key) external view returns (int256) {`
		`58    function getAddress(bytes32 key) external view returns (address) {`
		`67    function getData(bytes32 key) external view returns (bytes32) {`
		`76    function getBool(bytes32 key) external view returns (bool) {`
			349	Minor	Suboptimal		These functions are redundant as corresponding mappings are public.

DataStore.sol	17	`17        return uintValues[key];`	350	Minor	Suboptimal		Consider the usage of default NOT_SET value=0, and increment every stored value by 1, to distinguish between initialized and not initialized values.

DataStore.sol	22, 43, 64, 73, 82	`22        return value;`
		`43        return value;`
		`64        return value;`
		`73        return value;`
		`82        return value;`
			351	Minor	Suboptimal	Returning an argument as it doesn't make sense.	Consider returning the old value or returning nothing.

DataStore.sol	21, 26, 32, 42, 47, 53, 63, 72, 81	`21        uintValues[key] = value;`
		`26        uint256 nextUint = uintValues[key] + value;`
		`32        uint256 nextUint = uintValues[key] - value;`
		`42        intValues[key] = value;`
		`47        int256 nextInt = intValues[key] + value;`
		`53        int256 nextInt = intValues[key] - value;`
		`63        addressValues[key] = value;`
		`72        dataValues[key] = value;`
		`81        boolValues[key] = value;`
			352	Minor	Suboptimal		Consider emitting and event on important data updates.

DataStore.sol	25, 31, 46, 52	`25    function incrementUint(bytes32 key, uint256 value) external onlyController returns (uint256) {`
		`31    function decrementUint(bytes32 key, uint256 value) external onlyController returns (uint256) {`
		`46    function incrementInt(bytes32 key, int256 value) external onlyController returns (int256) {`
		`52    function decrementUint(bytes32 key, int256 value) external onlyController returns (int256) {`
			353	Minor	Bad naming	The words “increment” and “decrement” are usually associated with increasing and decreasing by one, while these functions increase and decrease by arbitrary value.	Consider using words “increase” and “decrease”.

File	Lines	Code	CVF	Severity	Category	Description	Recommendation

DataStore.sol	26–28, 47–49	26 uint256 nextUint = uintValues[key] + value;
		27 uintValues[key] = nextUint;
		28 return nextUint;
		47 int256 nextInt = intValues[key] + value;
		48 intValues[key] = nextInt;
		49 return nextInt;
			354	Minor	Suboptimal		This could be simplified as:
							return uintValues [key] += value;

DataStore.sol	32–34, 53–55	32 uint256 nextUint = uintValues[key] - value;
		33 uintValues[key] = nextUint;
		34 return nextUint;
		53 int256 nextInt = intValues[key] - value;
		54 intValues[key] = nextInt;
		55 return nextInt;
			355	Minor	Suboptimal		This could be simplified as:
							return uintValues [key] -= value;

Bank.sol	14, 51	14 constructor(RoleStore _roleStore) RoleModule(_roleStore) {}
		51 function _afterTransferOut(address /* token */) internal virtual {}
			356	Minor	Procedural		It is a good practice to put a comment into an empty block to explain why the block is empty.

Bank.sol	16, 22, 34, 51	16 function transferOut(address token, uint256 amount, address receiver) external onlyController {
		22 address token,
		34 function _transferOut(address token, uint256 amount, address receiver) internal {
		51 function _afterTransferOut(address /* token */) internal virtual {}
			357	Minor	Bad datatype		The type of the "token" argument should be "IERC20".

Bank.sol	21	21 address weth,	358	Minor	Bad datatype		The type of the "weth" argument should be "IWETH".
Bank.sol	21	21 address weth,	359	Minor	Procedural	The way of handling the WETH token is different than usual ERC20, it looks safer to set WETH address once as a immutable variable rather than receive it dynamically.	Consider initialization of WETH address once.
Bank.sol	25	25 bool hasCollateralInETH	360	Minor	Documentation		The semantic of the function is not clear, consider documenting how this argument is used and why does it affect the way how transferOut works.
Bank.sol	42	42 function _transferOutEth(address token, uint256 amount, address receiver) internal {	361	Minor	Bad datatype		The type of the "token" argument should be IWETH".
Bank.sol	46	46 payable(receiver).transfer(amount);	362	Minor	Suboptimal	Usage of the "transfer" function is discouraged.	Consider using "call" instead.
Bank.sol	51	51 function _afterTransferOut(address /* token */) internal virtual {}	363	Minor	Unclear behavior	This function should accept the amount and the recipient address as arguments.	
FeeUtils.sol	5	5library FeeUtils {	364	Minor	Bad naming	Despite the name, this library don't contains any utilities, but only certain constants.	Consider renaming.
Deposit.sol	6	6 struct Props {	365	Minor	Bad naming	The name is too generic.	Consider renaming to "DepositProps" or just "Deposit".