Report

v. 1.0

Customer

GMX

Smart Contract Audit

gmx-synthetics

20th November 2022

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	20.11.22	A. Zveryanskaya	Initial Draft
0.2	20.11.22	A. Zveryanskaya	Minor revision
1.0	20.11.22	A. Zveryanskaya	Release

2 Introduction

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

GMX is a decentralized spot and perpetual exchange that supports low swap fees and low price impact trades.

After fixing the indicated issues the smart contracts should be re-audited.

3 Project scope

We were asked to review:

- Original Code

Files:

bank/			
	Bank.sol	StrictBank.sol	
data/			
	Keys.sol	DataStore.sol	
deposit/			
	Deposit.sol	DepositStore.sol	DepositUtils.sol
eth/			
	EthUtils.sol	IWETH.sol	
exchange/			
	DepositHandler.sol	LiquidationHandler.sol	OrderHandler.sol
	WithdrawalHandler.sol		
feature/			
	FeatureUtils.sol		
fee/			
	FeeReceiver.sol	FeeUtils.sol	
gas/			
	GasUtils.sol		
gov/			
	Governable.sol		

market/			
	Market.sol	MarketFactory.sol	MarketToken.sol
	MarketUtils.sol	MarketStore.sol	
nonce/			
	NonceUtils.sol		
oracle/			
	IPriceFeed.sol	Oracle.sol	OracleModule.sol
	OracleStore.sol	OracleUtils.sol	
order/			
	DecreaseOrderUtils.sol	IncreaseOrderUtils.sol	LiquidationUtils.sol
	Order.sol	OrderStore.sol	OrderUtils.sol
	SwapOrderUtils.sol		
position/			
	DecreasePositionU-tils.sol	IncreasePositionUtils.sol	Position.sol
	PositionStore.sol	PositionUtils.sol	
pricing/			
	PositionPricingUtils.sol	PricingUtils.sol	SwapPricingUtils.sol
reader/			
	Reader.sol		
role/			
	Role.sol	RoleModule.sol	RoleStore.sol
router/			
	ExchangeRouter.sol	Router.sol	
swap/			
	SwapUtils.sol		
timelock/			
	Timelock.sol		
utils/			
	Array.sol	Bits.sol	Calc.sol
	EnumerableValues.sol	Precision.sol	
withdrawal/			
	Withdrawal.sol	WithdrawalStore.sol	WithdrawalUtils.sol

4 Methodology

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

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

We classify issues by the following severity levels:

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

5 Our findings

We found 1 critical, 37 major, and a few less important issues. All identified Critical and Major issues have been fixed.

6 Critical Issues

CVF-1 INFO

- **Category** Flaw
- **Source** MarketUtils.sol

Description A trader's profit results in a pool's loss, so the pending trader PnL should be subtracted rather than added.

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);
```

7 Major Issues

CVF-2 INFO

- **Category** Suboptimal
- **Source** SwapUtils.sol

Description Transferring fees on every atomic swap is inefficient.

Recommendation Consider accumulating fees in the pool and allowing the fee receiver to take them later.

84
```
PricingUtils.transferFees(
```

CVF-3 INFO

- **Category** Suboptimal
- **Source** ExchangeRouter.sol

Description This code should be executed only when msg.value is not zero.

125
```
address weth = EthUtils.weth(dataStore);
IWETH(weth).deposit{value: msg.value}();
IERC20(weth).safeTransfer(address(receiver), msg.value);
```

CVF-4 INFO

- **Category** Readability
- **Source** PositionUtils.sol

Recommendation This should be reordered as: if (position.sizeInUsd == sizeDeltaUsd) {...} else if (totalPositionPnl > 0) {...} else {...}

53
```
if (totalPositionPnl > 0) {
    sizeDeltaInTokens = position.sizeInTokens * sizeDeltaUsd /
        ↪ position.sizeInUsd;
} else {
    uint256 nextSizeInUsd = position.sizeInUsd - sizeDeltaUsd;
    uint256 nextSizeInTokens = position.sizeInTokens * nextSizeInUsd
        ↪  / position.sizeInUsd;
    sizeDeltaInTokens = position.sizeInTokens - nextSizeInTokens;
}
```

61
```
if (position.sizeInUsd == sizeDeltaUsd) {
    sizeDeltaInTokens = position.sizeInTokens;
}
```

CVF-5 INFO

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

Description This allows associating the same key with several accounts.

Recommendation 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); }

```
20 accountPositionKeys[account].add(key);
```

CVF-6 INFO

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

Description In case of incorrect “account” value, this line will not be able to remove the account to key association.

Recommendation Consider obtaining the account from the “positions” mapping rather than accepting as an argument.

```
26 accountPositionKeys[account].remove(key);
```

CVF-7 INFO

- **Category** Flaw
- **Source** IncreasePositionUtils.sol

Description Division by a scaled price could lead to precision degradation for cheap tokens with lots of decimals.

Recommendation Consider scaling up the denominator and then dividing by an unscaled price.

```
66 uint256 sizeDeltaInTokens = params.order.sizeDeltaUsd() / prices.
   ↪ indexTokenPrice;
```

CVF-8 INFO

- **Category** Suboptimal
- **Source** IncreasePositionUtils.sol

Description Sending out fees on every position change is inefficient.

Recommendation Consider accumulating fees in the pool and allowing the fee receiver to take them later.

```
140 PricingUtils.transferFees(
```

CVF-9 INFO

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

Description Phantom underflow is possible here.

Recommendation Consider doing the subtraction after the addition.

```
491 totalBorrowing -= prevPositionSizeInUsd *
       ↪ prevPositionBorrowingFactor;
    totalBorrowing += nextPositionSizeInUsd *
       ↪ nextPositionBorrowingFactor;
```

CVF-10 INFO

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

Description 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.

Recommendation 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.

```
25 marketTokens.remove(marketToken);
```

CVF-11 INFO

- **Category** Flaw
- **Source** GasUtils.sol

Description In case of an unsuccessful transfer, ether will be accumulated at the contract's balance.

Recommendation Consider implementing some way for the user to extract this ether later. For example, maintain a mapping of pending ether per user.

```
57 bool success = payable(user).send(refundFeeForUser);
```

CVF-12 INFO

- **Category** Flaw
- **Source** GasUtils.sol

Description Transaction gas price could be manipulated by a message sender.

Recommendation Consider using a reliable gas price oracle instead.

```
63 uint256 minExecutionFee = gasLimit * tx.gasprice;
```

CVF-13 INFO

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

Recommendation There is no restriction for account in createWithdrawal, consider adding additional check inside createWithdrawal (or use msg.sender)

```
106 require(withdrawal.account != address(0), "WithdrawalUtils: empty
    ↪ withdrawal");
```

```
204 require(withdrawal.account != address(0), "WithdrawalUtils: empty
    ↪ withdrawal");
```

CVF-14 INFO

- **Category** Suboptimal
- **Source** EnumerableValues.sol

Description Copying array elements one by one is inefficient.

Recommendation Consider using the "identity" precompile to copy a memory range directly from the array underneath the set.

17
```
for (uint256 i = start; i < end; i++) {
    items[i - start] = set.at(i);
}
```

30
```
for (uint256 i = start; i < end; i++) {
    items[i - start] = set.at(i);
}
```

43
```
for (uint256 i = start; i < end; i++) {
    items[i - start] = set.at(i);
}
```

CVF-15 INFO

- **Category** Suboptimal
- **Source** SwapPricingUtils.sol

Recommendation Three parameters here seem redundant. Two would be enough.

126
```
uint256 spreadFactor = dataStore.getUint(Keys.swapSpreadFactorKey(
    ↪ marketToken));
uint256 feeFactor = dataStore.getUint(Keys.swapFeeFactorKey(
    ↪ marketToken));
uint256 feeReceiverFactor = dataStore.getUint(feeReceiverFactorKey);
```

CVF-16 INFO

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

Description This formula is asymmetric. For example, it allows (long < short, nextLong = nextShort), but doesn't allow (long > short, nextLong = nextShort).

Recommendation Consider handling the equality cases properly.

```
62 bool isSameSideRebalance = openInterestParams.longOpenInterest <=
      ↪ openInterestParams.shortOpenInterest == openInterestParams.
      ↪ nextLongOpenInterest <= openInterestParams.
      ↪ nextShortOpenInterest;
```

CVF-17 INFO

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

Description This function always increases an open interest (either long or short) but never decreases it.

Recommendation Consider decreasing the other side open interest when possible.

```
90 function getNextOpenInterest(
```

CVF-18 INFO

- **Category** Suboptimal
- **Source** PositionPricingUtils.sol

Recommendation 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.

```
139 uint256 feeFactor = dataStore.getUint(Keys.positionFeeFactorKey(
       ↪ position.market));
140 uint256 feeReceiverFactor = dataStore.getUint(feeReceiverFactorKey);
    uint256 spreadFactor = dataStore.getUint(Keys.
       ↪ positionSpreadFactorKey(position.market));
```

CVF-19 INFO

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

Recommendation As zero account is used as a marker of a non-existing order, consider explicitly requiring the account to be not zero.

```
72 address account,
```

CVF-20 INFO

- **Category** Suboptimal
- **Source** OrderUtils.sol

Recommendation 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.

```
162 return orderType == Order.OrderType.MarketSwap ||
        orderType == Order.OrderType.MarketIncrease ||
        orderType == Order.OrderType.MarketDecrease;
```

```
168 return orderType == Order.OrderType.LimitSwap ||
        orderType == Order.OrderType.LimitIncrease ||
170     orderType == Order.OrderType.LimitDecrease;
```

```
174 return orderType == Order.OrderType.MarketSwap ||
        orderType == Order.OrderType.LimitSwap;
```

```
179 return orderType == Order.OrderType.MarketIncrease ||
180     orderType == Order.OrderType.LimitIncrease;
```

```
184 return orderType == Order.OrderType.MarketIncrease ||
        orderType == Order.OrderType.LimitIncrease;
```

```
189 return orderType == Order.OrderType.MarketDecrease ||
190     orderType == Order.OrderType.LimitDecrease ||
        orderType == Order.OrderType.StopLossDecrease;
```

```
207 if (orderType == Order.OrderType.MarketIncrease || orderType ==
      ↪ Order.OrderType.MarketDecrease) {
```

CVF-21 INFO

- **Category** Flaw
- **Source** Timelock.sol

Description 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.

Recommendation Consider adding appropriate length checks.

```
43 function fastSetUints(DataStore dataStore, string[] memory prefixes,
   ↪  bytes[] memory data, uint256[] memory values) external
   ↪ onlyAdmin {
```

CVF-22 INFO

- **Category** Suboptimal
- **Source** RoleStore.sol

Recommendation It would be more efficient to do like this: if (roleMembers [key].add (account)) roles.add (key);

```
19 roles.add(key);
20 roleMembers[key].add(account);
```

CVF-23 INFO

- **Category** Suboptimal
- **Source** SwapOrderUtils.sol

Description This function should emit some event.

```
12 function processOrder(OrderUtils.ExecuteOrderParams memory params)
   ↪ external {
```

CVF-24 INFO

- **Category** Flaw
- **Source** SwapOrderUtils.sol

Description There is no explicit check to ensure that the swap path is not empty.

Recommendation Consider adding such a check.

```
17 address firstMarket = params.order.swapPath()[0];
```

CVF-25 INFO

- **Category** Suboptimal
- **Source** SwapOrderUtils.sol

Recommendation This should be “order.account()” to make the separate “transferOut” call unnecessary.

```
41 address(0)
```

CVF-26 INFO

- **Category** Suboptimal
- **Source** MarketFactory.sol

Description This function should emit some event.

```
19 function createMarket(
```

CVF-27 INFO

- **Category** Procedural
- **Source** DecreasePositionUtils.sol

Recommendation This check should be moved into the “processCollateral” function. Having it here is error-prone.

```
73 if (values.remainingCollateralAmount < 0) {
       revert("Insufficient␣collateral");
   }
```

CVF-28 INFO

- **Category** Flaw
- **Source** DecreasePositionUtils.sol

Description The remaining collateral may go negative here.

Recommendation Consider adding an explicit check to prevent this.

```
196 remainingCollateralAmount += values.realizedPnlAmount;
```

```
219 remainingCollateralAmount += fees.totalNetCostAmount;
```

CVF-29 INFO

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

Description 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.

Recommendation Consider using the “muldiv” function as described here: https://xn–2-umb.com/21/muldiv/

```
16 return amount * factor / FLOAT_PRECISION;
```

```
20 return amount.toInt256() * factor / FLOAT_PRECISION.toInt256();
```

CVF-30 INFO

- **Category** Procedural
- **Source** Role.sol

Description Public constants don’t make much sense in a library.

Recommendation Consider changing the acces level to “internal”.

```
6  bytes32 public constant CONTROLLER = keccak256("CONTROLLER");
   bytes32 public constant ROUTER_PLUGIN = keccak256("ROUTER_PLUGIN");
   bytes32 public constant MARKET_KEEPER = keccak256("MARKET_KEEPER");
   bytes32 public constant ORDER_KEEPER = keccak256("ORDER_KEEPER");
10 bytes32 public constant PRICING_KEEPER = keccak256("PRICING_KEEPER")
      ↪ ;
   bytes32 public constant LIQUIDATION_KEEPER = keccak256("
      ↪ LIQUIDATION_KEEPER");
```

CVF-31 INFO

- **Category** Suboptimal
- **Source** FeeReceiver.sol

Description The function is declared as external so anyone can call it and trigger backend on false event.

Recommendation Consider restricting the caller or include emitting inside the usage code.

8
```
function notifyFeeReceived(bytes32 key, address token, uint256
   ↪ amount) external {
```

CVF-32 INFO

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

Description There is no separate function to cancel a withdrawal.

Recommendation Consider adding such a function.

22
```
contract WithdrawalHandler is RoleModule, ReentrancyGuard,
   ↪ OracleModule {
```

CVF-33 INFO

- **Category** Suboptimal
- **Source** WithdrawalHandler.sol

Description These functions should emit some events.

49
```
function createWithdrawal(
```

82
```
function executeWithdrawal(
```

CVF-34 INFO

- **Category** Suboptimal
- **Source** DepositUtils.sol

Recommendation As zero account is used as as a marker of a non-existing deposit, consider explicitly requiring the account to bo non-zero.

```
78 params.account,
```

CVF-35 INFO

- **Category** Suboptimal
- **Source** LiquidationHandler.sol

Description This function should emit some event.

```
46 function liquidatePosition(
```

CVF-36 INFO

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

Recommendation Confused naming may lead to misusing the function, consider renaming to decrementInt

```
52 function decrementUint(bytes32 key, int256 value) external
   ↪ onlyController returns (int256) {
```

CVF-37 INFO

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

Description This formula is asymmetric. For example, it allows (A = B, nextA < nextB) but doesn't allow (A = B, nextA > nextB).

Recommendation Consider handling the A = B and nextA = nextB cases properly.

```
68 bool isSameSideRebalance = poolParams.poolUsdForTokenA <= poolParams
   ↪ .poolUsdForTokenB == poolParams.nextPoolUsdForTokenA <=
   ↪ poolParams.nextPoolUsdForTokenB;
```

CVF-220 INFO

- **Category** Flaw
- **Source** RoleStore.sol

Description This function doesn't remove roles with no members from the "roles" set.

Recommendation Consider refactoring like this: EnumerableSet.Bytes32Set storage members = roleMembers [key]; if (members.remove (account) && members.length () == 0) roles.remove (key);

```
23 function revokeRole(address account, bytes32 key) external onlyGov {
```

8 Moderate Issues

CVF-38 INFO

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

Description Here should be some unique prefix hashed to guaranteed key uniqueness.

```
75 bytes32 key = keccak256(abi.encodePacked(account, market,
   ↪ collateralToken, isLong));
```

CVF-39 INFO

- **Category** Flaw
- **Source** Oracle.sol

Description block.chainid may change in future or in forks. The ”freezing” the chainid allows replay attack on forks.

Recommendation Consider using CACHED chain_id as it is done in EIP712.sol. Also check the discusssion here.

```
79 SALT = keccak256(abi.encodePacked(block.chainid, "xget-oracle-v1"));
```

CVF-40 INFO

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

Description 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.

Recommendation Consider scaling a product, rather than a price.

```
199 int256 openInterestValue = (openInterestInTokens * indexTokenPrice).
   ↪ toInt256();
```

```
345     reservedUsd = openInterestInTokens * prices.indexTokenPrice;
```

```
541 uint256 poolUsd = poolAmount * poolTokenPrice;
```

CVF-41 INFO

- **Category** Flaw
- **Source** MarketStore.sol

Recommendation Consider checking if market is empty or not, to be sure that specific market was not already removed.

```
33 return markets[marketToken];
```

CVF-42 INFO

- **Category** Flaw
- **Source** GasUtils.sol

Description 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.

Recommendation Consider using a reliable gas price oracle such as chainlink, instead of just the gas price of the current transaction.

```
39 uint256 executionFeeForKeeper = adjustGasLimit(dataStore, gasUsed) *
   ↪  tx.gasprice;
```

CVF-43 INFO

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

Description This line throws in case start > end. The caller cannot efficiently prevent this, as the “end” value could be adjusted in the previous line.

Recommendation Consider returning an empty array in case start >= end here.

```
16 bytes32[] memory items = new bytes32[](end - start);
```

```
29 address[] memory items = new address[](end - start);
```

```
42 uint256[] memory items = new uint256[](end - start);
```

CVF-44 INFO

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

Description This conditions is always false here, as the “orderType == Order.OrderType.LimitIncrease” case is handled by tyhe previous conditional statement.

```
307 orderType == Order.OrderType.LimitIncrease ||
```

CVF-45 INFO

- **Category** Procedural
- **Source** OracleModule.sol

Description Emitting an event before reverting a transaction doesn’t make sense, as the event will be reverted as well.

Recommendation Consider returning instead of reverting.

```
18 emit OracleError(reason);
   revert(Keys.ORACLE_ERROR);
```

CVF-46 INFO

- **Category** Suboptimal
- **Source** OrderHandler.sol

Description 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.

```
130 function cancelOrder(bytes32 key) external {
```

CVF-47 INFO

- **Category** Suboptimal
- **Source** DepositUtils.sol

Description For cheap tokens with lots of decimals, using a scaled price could lead to precision degradation.

Recommendation Consider multiplying by an unscaled price and scaling the product.

```
113 uint256 longTokenUsd = deposit.longTokenAmount * longTokenPrice;
    uint256 shortTokenUsd = deposit.shortTokenAmount * shortTokenPrice;
```

```
126         (deposit.longTokenAmount * longTokenPrice).toInt256(),
            (deposit.shortTokenAmount * shortTokenPrice).toInt256()
```

CVF-48 INFO

- **Category** Suboptimal
- **Source** Keys.sol

Description encodePacked function is not injective, so that it is possible that different inputs concatenate to the same output of the function.

Recommendation 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.

93
```
return keccak256(abi.encodePacked(
    DEPOSIT_GAS_LIMIT,
    singleToken
));
```

101
```
return keccak256(abi.encodePacked(
    WITHDRAWAL_GAS_LIMIT,
    singleToken
));
```

132
```
return keccak256(abi.encodePacked(
    CREATE_DEPOSIT_FEATURE,
    module
));
```

139
140
```
return keccak256(abi.encodePacked(
    EXECUTE_DEPOSIT_FEATURE,
    module
));
```

146
```
return keccak256(abi.encodePacked(
    CREATE_WITHDRAWAL_FEATURE,
    module
));
```

153
```
return keccak256(abi.encodePacked(
    EXECUTE_WITHDRAWAL_FEATURE,
    module
));
```

160
```
return keccak256(abi.encodePacked(
    CREATE_ORDER_FEATURE,
    module,
    orderType
));
```

```
168 return keccak256(abi.encodePacked(
        EXECUTE_ORDER_FEATURE,
170     module,
        orderType
    ));

176 return keccak256(abi.encodePacked(
        UPDATE_ORDER_FEATURE,
        module,
        orderType
180 ));

184 return keccak256(abi.encodePacked(
        CANCEL_ORDER_FEATURE,
        module,
        orderType
    ));

192 return keccak256(abi.encodePacked(
        LIQUIDATE_POSITION_FEATURE,
        module
    ));

199 return keccak256(abi.encodePacked(
200     POSITION_IMPACT_FACTOR,
        market,
        isPositive
    ));

207 return keccak256(abi.encodePacked(
        POSITION_IMPACT_EXPONENT_FACTOR,
        market
210 ));

214 return keccak256(abi.encodePacked(
        POSITION_SPREAD_FACTOR,
        market
    ));

221 return keccak256(abi.encodePacked(
        POSITION_FEE_FACTOR,
        market
    ));
```

```
228 return keccak256(abi.encodePacked(
        SWAP_IMPACT_FACTOR,
230     market,
        isPositive
    ));
```

```
236 return keccak256(abi.encodePacked(
        SWAP_IMPACT_EXPONENT_FACTOR,
        market
    ));
```

```
244 return keccak256(abi.encodePacked(
        SWAP_SPREAD_FACTOR,
        market
    ));
```

```
251 return keccak256(abi.encodePacked(
        SWAP_FEE_FACTOR,
        market
    ));
```

```
258 return keccak256(abi.encodePacked(
        ORACLE_PRECISION,
260     token
    ));
```

```
266 return keccak256(abi.encodePacked(
        OPEN_INTEREST,
        market,
        isLong
270 ));
```

```
275 return keccak256(abi.encodePacked(
        OPEN_INTEREST_IN_TOKENS,
        market,
        isLong
    ));
```

```
284 return keccak256(abi.encodePacked(
        COLLATERAL_SUM,
        market,
        collateralToken,
        isLong
    ));
```

```
294 return keccak256(abi.encodePacked(
        POOL_AMOUNT,
        market,
        token
    ));
```

```
303 return keccak256(abi.encodePacked(
        SWAP_IMPACT_POOL_AMOUNT,
        market,
        token
    ));
```

```
311 return keccak256(abi.encodePacked(
        RESERVE_FACTOR,
        market,
        isLong
    ));
```

```
319 return keccak256(abi.encodePacked(
320     FUNDING_FACTOR,
        market
    ));
```

```
326 return keccak256(abi.encodePacked(
        CUMULATIVE_FUNDING_FACTOR,
        market,
        isLong
330 ));
```

```
334 return keccak256(abi.encodePacked(
        CUMULATIVE_FUNDING_FACTOR_UPDATED_AT,
        market
    ));
```

```
341 return keccak256(abi.encodePacked(
        BORROWING_FACTOR,
        market,
        isLong
    ));
```

```
349 return keccak256(abi.encodePacked(
350     CUMULATIVE_BORROWING_FACTOR,
        market,
        isLong
    ));
```

```
357 return keccak256(abi.encodePacked(
        CUMULATIVE_BORROWING_FACTOR_UPDATED_AT,
        market,
360     isLong
    ));
```

```
365 return keccak256(abi.encodePacked(
        TOTAL_BORROWING,
        market,
        isLong
    ));
```

```
373 return keccak256(abi.encodePacked(
        PRICE_FEED,
        token
    ));
```

```
380 return keccak256(abi.encodePacked(
        PRICE_FEED_PRECISION,
        token
    ));
```

```
387 return keccak256(abi.encodePacked(
        STABLE_PRICE,
        token
390 ));
```

9 Minor Issues

CVF-49 INFO

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

Description It's not clear why to use hash of uint256 as a key.

Recommendation Consider the usage of just uint256 nonce as a key to increase readability and avoid redundant hashing.

```
12 mapping(bytes32 => Withdrawal.Props) public withdrawals;
```

CVF-50 INFO

- **Category** Procedural
- **Source** WithdrawalStore.sol

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

```
15 constructor(RoleStore _roleStore) StrictBank(_roleStore) {}
```

CVF-51 INFO

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

Description 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".

Recommendation Consider also the name "store".

```
17 function set(bytes32 key, Withdrawal.Props memory withdrawal)
     ↪ external onlyController {
```

CVF-52 INFO

- **Category** Suboptimal
- **Source** WithdrawalStore.sol

Description An event should be emitted here

```
19 withdrawalKeys.add(key);
```

CVF-53 INFO

- **Category** Suboptimal
- **Source** Array.sol

Recommendation This function would be more useful if it would accept the default value as an argument.

```
6 function get(bytes32[] memory arr, uint256 index) internal pure
    ↪ returns (bytes32) {
```

CVF-54 INFO

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

Description The name is too generic.

Recommendation Consider making it more specific to emphasize the ability of this function to handle invalid indexes.

```
6 function get(bytes32[] memory arr, uint256 index) internal pure
    ↪ returns (bytes32) {
```

CVF-55 INFO

- **Category** Suboptimal
- **Source** Array.sol

Recommendation 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

```
21 newArr[index] = value;
```

CVF-56 INFO

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

Description Confusing function name as no copy is created.

Recommendation Consider renaming

```
15 if (index < arr.length) {
       arr[index] = value;
       return arr;
   }
```

CVF-57 INFO

- **Category** Suboptimal
- **Source** Array.sol

Description Increasing the array size to index + 1 is suboptimal, as it may lead to O(n^2) complexity for sequential writes.

Recommendation Consider increasing the array size to index * 3 / 2 + 1 or something like this.

```
20 bytes32[] memory newArr = createResized(arr, index + 1);
```

CVF-58 INFO

- **Category** Documentation
- **Source** Array.sol

Recommendation Nothing is created here, consider documenting it.

```
27 if (length <= arr.length) {
       return arr;
   }
```

CVF-59 INFO

- **Category** Suboptimal
- **Source** Array.sol

Description In a quite common case when arr + arr.length equals to the free memory pointer, it is possible to resize the array without copying.

Recommendation Consider implementing such logic.

```
31 bytes32[] memory newArr = new bytes32[](length);
```

CVF-60 INFO

- **Category** Suboptimal
- **Source** Array.sol

Description Copying array elements only by one is suboptimal.

Recommendation Consider using the identity precompile.

```
33 for (uint256 i = 0; i < arr.length; i++) {
       newArr[i] = arr[i];
   }
```

CVF-61 INFO

- **Category** Suboptimal
- **Source** Array.sol

Recommendation Use unchecked declaration for i++ to save gas.

```
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++) {
```

CVF-62 INFO

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

Description This function works correctly only for sorted arrays.

Recommendation Consider reflecting this fact in the function name or in a documentation comment.

```
60 function getMedian(uint256[] memory arr) internal pure returns (
   ↪ uint256) {
```

CVF-63 INFO

- **Category** Flaw
- **Source** Array.sol

Recommendation The function will fail if length=0, consider adding a special check for this case.

```
60 function getMedian(uint256[] memory arr) internal pure returns (
   ↪ uint256) {
    if (arr.length % 2 == 1) {
        return arr[arr.length / 2];
    }

    return (arr[arr.length / 2] + arr[arr.length / 2 - 1]) / 2;
}
```

CVF-64 INFO

- **Category** Suboptimal
- **Source** Array.sol

Recommendation Right shift would be more efficient that division.

```
65 return (arr[arr.length / 2] + arr[arr.length / 2 - 1]) / 2;
```

CVF-65 INFO

- **Category** Suboptimal
- **Source** Array.sol

Description The expression “arr.length / 2” is calculated twice.

Recommendation Consider calculating once and reusing.

```
65 return (arr[arr.length / 2] + arr[arr.length / 2 - 1]) / 2;
```

CVF-66 INFO

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

Description 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.

```
65 return (arr[arr.length / 2] + arr[arr.length / 2 - 1]) / 2;
```

CVF-67 INFO

- **Category** Suboptimal
- **Source** SwapUtils.sol

Description Having two structs with very similar names is confusing.

Recommendation Consider using more distinct names.

```
12 struct SwapParams {
```

```
23 struct _SwapParams {
```

CVF-68 INFO

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

Recommendation The type of these fields could be more specific.

```
16 address tokenIn;
```

```
25 address tokenIn;
```

```
31 address tokenOut;
```

CVF-69 INFO

- **Category** Suboptimal
- **Source** SwapUtils.sol

Recommendation Consider declaring return values in the function declaration or use NatSpec.

```
39 // returns tokenOut, outputAmount
```

CVF-70 INFO

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

Description These variable names are confusing, as they used as both, input and output.

Recommendation Consider renaming to just “token” and “amount“, or “currentToken” and “currentAmount”.

```
41 address tokenOut = params.tokenIn;
   uint256 outputAmount = params.amountIn;
```

CVF-71 INFO

- **Category** Suboptimal
- **Source** SwapUtils.sol

Description

Recommendation Consider using unchecked declaration to save gas.

```
44  for (uint256 i = 0; i < params.markets.length; i++) {
        Market.Props memory market = params.markets[i];
        uint256 nextIndex = i + 1;
        address receiver;
        if (nextIndex < params.markets.length) {
            receiver = params.markets[nextIndex].marketToken;
50      } else {
            receiver = params.receiver;
        }

        _SwapParams memory _params = _SwapParams(
            market,
            tokenOut,
            outputAmount,
            receiver
        );
60      (tokenOut, outputAmount) = _swap(params, _params);
    }
```

CVF-72 INFO

- **Category** Suboptimal
- **Source** SwapUtils.sol

Description The expression “fees.amountAfterFees * cache.tokenInPrice” is calculated twice.

Recommendation Consider calculating once and reusing.

```
99  (fees.amountAfterFees * cache.tokenInPrice).toInt256(),
100 -(fees.amountAfterFees * cache.tokenInPrice).toInt256()
```

CVF-73 INFO

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

Description Phantom overflow is possible here.

Recommendation Consider using the “muldiv” function.

```
105 cache.amountOut = cache.amountIn * cache.tokenInPrice / cache.
    ↪ tokenOutPrice;
```

```
137 cache.amountOut = cache.amountIn * cache.tokenInPrice / cache.
    ↪ tokenOutPrice;
```

CVF-74 INFO

- **Category** Procedural
- **Source** ExchangeRouter.sol

Recommendation These variables should be declared as immutable.

```
19 Router public router;
20 DataStore public dataStore;
   DepositHandler public depositHandler;
   WithdrawalHandler public withdrawalHandler;
   OrderHandler public orderHandler;
   DepositStore public depositStore;
   WithdrawalStore public withdrawalStore;
   OrderStore public orderStore;
```

CVF-75 INFO

- **Category** Suboptimal
- **Source** ExchangeRouter.sol

Recommendation This variable is redundant, as “msg.sender” is cheaper to read than a local variable.

```
60 address account = msg.sender;
```

```
90 address account = msg.sender;
```

```
110 address account = msg.sender;
```

CVF-76 INFO

- **Category** Procedural
- **Source** ExchangeRouter.sol

Recommendation Consider using named arguments

```
94  return withdrawalHandler.createWithdrawal(
        account,
        market,
        marketTokensLongAmount,
        marketTokensShortAmount,
        minLongTokenAmount,
100     minShortTokenAmount,
        hasCollateralInETH,
        executionFee
    );
```

CVF-77 INFO

- **Category** Documentation
- **Source** PositionUtils.sol

Description The semantics of the returned values is unclear.

Recommendation Consider giving descriptive names to the returned values and/or adding a documentation comment.

```
28  ) internal pure returns (int256, uint256) {
```

CVF-78 INFO

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

Recommendation Consider giving these names to the returned values, rather then specifying them in a comment.

```
38  // returns (positionPnlUsd, sizeDeltaInTokens)
```

CVF-79 INFO

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

Description 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.

Recommendation Consider renaming.

```
38 // returns (positionPnlUsd, sizeDeltaInTokens)
```

CVF-80 INFO

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

Description Phantom overflow is possible here.

Recommendation Consider using the “muldiv” function.

```
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) {
```

CVF-81 INFO

- **Category** Suboptimal
- **Source** PositionUtils.sol

Recommendation There are simpler ways to divide with rounding up. For example: (x + y - 1) / y

```
56 uint256 nextSizeInUsd = position.sizeInUsd - sizeDeltaUsd;
   uint256 nextSizeInTokens = position.sizeInTokens * nextSizeInUsd /
      ↪ position.sizeInUsd;
   sizeDeltaInTokens = position.sizeInTokens - nextSizeInTokens;
```

CVF-82 INFO

- **Category** Documentation
- **Source** PositionUtils.sol

Description The difference between “totalPositionPnl” and “positionPnlUsd” is unclear.

Recommendation Consider documenting.

```
65 int256 positionPnlUsd = totalPositionPnl * sizeDeltaInTokens.
   ↪ toInt256() / position.sizeInTokens.toInt256();
```

CVF-83 INFO

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

Recommendation The type of the “collateralToken” argument could be more specific.

```
74 function getPositionKey(address account, address market, address
   ↪ collateralToken, bool isLong) internal pure returns (bytes32)
   ↪ {
```

CVF-84 INFO

- **Category** Suboptimal
- **Source** PositionUtils.sol

Recommendation This could be optimized as: if (position.sizeInUsd | position.sizeInTokens | position.collateralAmount == 0)

```
80 if (position.sizeInUsd == 0 || position.sizeInTokens == 0 ||
   ↪ position.collateralAmount == 0) {
```

CVF-85 INFO

- **Category** Readability
- **Source** PositionUtils.sol

Recommendation Use named args to avoid argument misusing.

```
125 PositionPricingUtils.GetPositionPricingParams(
        dataStore,
        market.marketToken,
        market.longToken,
        market.shortToken,
130     prices.longTokenPrice,
        prices.shortTokenPrice,
        -position.sizeInUsd.toInt256(),
        position.isLong
    )
```

CVF-86 INFO

- **Category** Suboptimal
- **Source** PositionUtils.sol

Recommendation This could be simplified as: return remainingCollateralUsd < minCollateralUsd || position.sizeInUsd * precision.FLOAT_PRECISION / remainingCollateralUsd.toUint256() > maxLeverage;

```
149 if (remainingCollateralUsd < minCollateralUsd) {
150     return true;
    }
```

```
154 if (position.sizeInUsd * Precision.FLOAT_PRECISION /
       ↪ remainingCollateralUsd.toUint256() > maxLeverage) {
        return true;
    }
```

```
158 return false;
```

CVF-87 INFO

- **Category** Procedural
- **Source** PositionUtils.sol

Description This function is never used and is too simple to be extracted.

Recommendation Consider removing this function.

```
161 function revertUnexpectedPositionState() internal pure {
```

CVF-88 INFO

- **Category** Documentation
- **Source** PositionStore.sol

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

```
16 constructor(RoleStore _roleStore) RoleModule(_roleStore) {}
```

CVF-89 INFO

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

Recommendation 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”.

```
18 function set(bytes32 key, address account, Position.Props memory
     ↪ position) external onlyController {
```

CVF-90 INFO

- **Category** Suboptimal
- **Source** PositionStore.sol

Description An event should be emitted here.

```
21 positionKeys.add(key);
```

CVF-91 INFO

- **Category** Suboptimal
- **Source** PositionStore.sol

Description An event should be emitted here.

```
27 positionKeys.remove(key);
```

CVF-92 INFO

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

Recommendation For completeness, consider implementing a version of this function with two arguments: “key” and “account”.

```
50 function contains(bytes32 key) public view returns (bool) {
```

CVF-93 INFO

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

Recommendation The type of this field could be more specific.

```
32 address collateralToken;
```

CVF-94 INFO

- **Category** Documentation
- **Source** OrderStore.sol

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

```
18 constructor(RoleStore _roleStore) StrictBank(_roleStore) {}
```

CVF-95 INFO

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

Description 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”.

Recommendation Consider also the name “store”.

```
20 function set(bytes32 key, Order.Props memory order) external
   ↪ onlyController {
```

CVF-96 INFO

- **Category** Procedural
- **Source** OrderStore.sol

Description The previous association for the same key is not removed here, so the same order key could be associated with several accounts.

Recommendation Consider ether forbidding adding the same order key several times or removing the existing association if any.

```
22 accountOrderKeys[order.account()].add(key);
```

CVF-97 INFO

- **Category** Suboptimal
- **Source** OrderStore.sol

Description Here an event should be emitted

```
23 orderKeys.add(key);
```

CVF-98 INFO

- **Category** Suboptimal
- **Source** OrderStore.sol

Recommendation The “account” argument is redundant as it could be derived as: orders[key].account()

```
26 function remove(bytes32 key, address account) external
   ↪ onlyController {
```

CVF-99 INFO

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

Description In case of incorrect “account” value, this line wouldn’t remove the association.

Recommendation Consider either requiring the association to exist, or deriving an account from the key.

```
28 accountOrderKeys[account].remove(key);
```

CVF-100 INFO

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

Description Here an event should be emitted

```
29 orderKeys.remove(key);
```

CVF-101 INFO

- **Category** Procedural
- **Source** IncreaseOrderUtils.sol

Recommendation This check should be done earlier.

```
12 MarketUtils.validateNonEmptyMarket(params.market);
```

CVF-102 INFO

- **Category** Readability
- **Source** IncreaseOrderUtils.sol

Recommendation Consider using named args.

```
(address collateralToken, uint256 collateralDeltaAmount) = SwapUtils
    ↪ .swap(SwapUtils.SwapParams(
    params.dataStore,
    params.oracle,
    params.feeReceiver,
    params.order.initialCollateralToken(),
    params.order.initialCollateralDeltaAmount(),
    params.swapPathMarkets,
    params.order.minOutputAmount(),
    address(0)
));
```

```
    IncreasePositionUtils.IncreasePositionParams(
        params.dataStore,
        params.positionStore,
        params.oracle,
        params.feeReceiver,
        params.market,
        params.order,
        position,
        positionKey,
        collateralToken,
        collateralDeltaAmount
    )
```

CVF-103 INFO

- **Category** Suboptimal
- **Source** IncreaseOrderUtils.sol

Recommendation This check should be done earlier.

```
if (position.market != address(0) || position.collateralToken !=
    ↪ address(0)) {
    PositionUtils.revertUnexpectedPositionState();
}
```

CVF-104 INFO

- **Category** Suboptimal
- **Source** IncreaseOrderUtils.sol

Recommendation This check should be done earlier.

```
47 if (collateralToken != params.market.longToken && collateralToken !=
   ↪  params.market.shortToken) {
    revert("OrderUtils: invalid collateralToken");
}
```

CVF-105 INFO

- **Category** Suboptimal
- **Source** DecreaseOrderUtils.sol

Recommendation Here “params.order” could be replaced with just “order”.

```
20     params.order.orderType(),
       params.order.updatedAtBlock(),
```

```
35         params.order.sizeDeltaUsd(),
```

```
40 if (adjustedSizeDeltaUsd == params.order.sizeDeltaUsd()) {
       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(),
           params.order.initialCollateralDeltaAmount(),
```

```
65         params.order.minOutputAmount(),
           params.order.account()
```

CVF-106 INFO

- **Category** Readability
- **Source** DecreaseOrderUtils.sol

Recommendation Consider using named args.

```
26  DecreasePositionUtils.DecreasePositionParams(
        params.dataStore,
        params.positionStore,
        params.oracle,
30      params.feeReceiver,
        params.market,
        order,
        position,
        positionKey,
        params.order.sizeDeltaUsd(),
        forLiquidation
    )
```

```
58  SwapUtils.swap(SwapUtils.SwapParams(
        params.dataStore,
60      params.oracle,
        params.feeReceiver,
        params.order.initialCollateralToken(),
        params.order.initialCollateralDeltaAmount(),
        params.swapPathMarkets,
        params.order.minOutputAmount(),
        params.order.account()
    ));
```

CVF-107 INFO

- **Category** Suboptimal
- **Source** DecreaseOrderUtils.sol

Recommendation 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.

```
49  if (order.swapPath().length == 0) {
```

CVF-108 INFO

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

Description Is 32 bits really enough for all token prices?

```
18 uint256 public constant COMPACTED_PRICE_LENGTH = 32;
```

CVF-109 INFO

- **Category** Suboptimal
- **Source** OracleUtils.sol

Recommendation Shift would be more efficient here.

```
32 uint256 slotIndex = index / COMPACTED_PRICES_PER_SLOT;
```

```
53 uint256 slotIndex = index / COMPACTED_BLOCK_NUMBERS_PER_SLOT;
```

CVF-110 INFO

- **Category** Suboptimal
- **Source** OracleUtils.sol

Recommendation Remainder and shift would be more efficient here.

```
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;
```

CVF-111 INFO

- **Category** Procedural
- **Source** OracleStore.sol

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

```
13 constructor(RoleStore _roleStore) RoleModule(_roleStore) {}
```

CVF-112 INFO

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

Description An event should be emitted here.

```
16 signers.add(account);
```

```
20 signers.remove(account);
```

CVF-113 INFO

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

Description Starting a structure name with an underscore (“_”) looks odd.

```
26 struct _SetPricesCache {
```

CVF-114 INFO

- **Category** Suboptimal
- **Source** Oracle.sol

Description Redundant data since we already know the blocknumber.

```
30 bytes32 blockHash;
```

CVF-115 INFO

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

Recommendation The type of this field could be more specific.

```
31 address token;
```

CVF-116 INFO

- **Category** Documentation
- **Source** Oracle.sol

Description The number format of this field is unclear.

Recommendation Consider documenting.

```
32 uint256 prevPrice;
```

CVF-117 INFO

- **Category** Documentation
- **Source** Oracle.sol

Description This equals to 15 which makes Oracle system not-scaleable for hundreds of signers.

Recommendation Consider adding a comment why 15 signers is enough.

```
41 uint256 public constant MAX_SIGNERS = 256 / SIGNER_INDEX_LENGTH - 1;
```

CVF-118 INFO

- **Category** Documentation
- **Source** Oracle.sol

Description In fact the MAX value is 255, 256 is already incorrect.

Recommendation Consider using 255.

```
42 // signer indexes are recorded in a signerIndexFlags uint256 value
   ↪ to check for uniqueness
```

CVF-119 INFO

- **Category** Suboptimal
- **Source** Oracle.sol

Recommendation This variable should be declared as “Immutable”.

```
45 OracleStore public oracleStore;
```

CVF-120 INFO

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

Description It’s not clear why is it neccessary to reset prices in storage.

Recommendation Consider doing that in memory scope.

```
50 EnumerableSet.AddressSet internal tempTokens;
```

CVF-121 INFO

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

Description In fact secondary price mapping does not hold second received price but it holds the last price.

Recommendation Consider renaming to firstPrices, lastPrices.

```
54 // the second occurrence will be stored in secondaryPrices
   mapping(address => uint256) public primaryPrices;
   mapping(address => uint256) public secondaryPrices;
```

CVF-122 INFO

- **Category** Suboptimal
- **Source** Oracle.sol

Description String error messages are suboptimal.

Recommendation Consider using named errors instead.

```
83 require(tempTokens.length() == 0, "Oracle: tempTokens not cleared");
```

CVF-123 INFO

- **Category** Suboptimal
- **Source** Oracle.sol

Recommendation These checks should be performed with the signes count value before allocating an array.

```
90 if (signers.length < dataStore.getUint(Keys.MIN_ORACLE_SIGNERS)) {
```

```
94 if (signers.length > MAX_SIGNERS) {
```

CVF-124 INFO

- **Category** Suboptimal
- **Source** Oracle.sol

Description The expression “dataStore.getUint(Keys.MIN_ORACLE_SIGNERS)” is calculated twice.

Recommendation Consider calculating once and reusing.

```
90 if (signers.length < dataStore.getUint(Keys.MIN_ORACLE_SIGNERS)) {
       revert MinOracleSigners(signers.length, dataStore.getUint(Keys.
          ↪ MIN_ORACLE_SIGNERS));
```

CVF-125 INFO

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

Description Probably an event should be emitted here.

```
131 secondaryPrices[token] = price;
```

CVF-126 INFO

- **Category** Suboptimal
- **Source** Oracle.sol

Description

Recommendation Consider replacing with ”unchecked {i+=1;}”

```
136 for (uint256 i = 0; i < length; i++) {
```

CVF-127 INFO

- **Category** Suboptimal
- **Source** Oracle.sol

Recommendation It should be cheaper to remove the last element from the set to avoid storage item swap.

```
137 address token = tempTokens.at(0);
```

CVF-128 INFO

- **Category** Suboptimal
- **Source** Oracle.sol

Description Using different precisions for different tokens make code more complicated.

Recommendation Consider using the same precision for all tokens.

```
210 function getPrecision(DataStore dataStore, address token) public
    ↪ view returns (uint256) {
```

CVF-129 INFO

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

Description There is no validation on alignment of arrays size.

```
216 address[] memory signers,
    address[] memory tokens,
    uint256[] memory compactedOracleBlockNumbers,
    uint256[] memory compactedPrices,
220 bytes[] memory signatures
```

CVF-130 INFO

- **Category** Procedural
- **Source** Oracle.sol

Description Note, that according to the Solidity docs you can only access blockhash for the last 256 blocks.

Recommendation Consider adding validation of minBlockConfirmations.

```
245 cache.blockHash = blockhash(cache.oracleBlockNumber);
```

CVF-131 INFO

- **Category** Suboptimal
- **Source** Oracle.sol

Description The expression “i * signers.length” is calculated on every loop iteration.

Recommendation Consider calculating once before the loop.

```
253 cache.priceAndSignatureIndex = i * signers.length + j;
```

CVF-132 INFO

- **Category** Suboptimal
- **Source** Oracle.sol

Recommendation 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.

```
270     prices[j] = price;
```

```
251 uint256[] memory prices = new uint256[](signers.length);
```

CVF-133 INFO

- **Category** Suboptimal
- **Source** Oracle.sol

Recommendation Probably an event should be emitted here.

```
275 secondaryPrices[cache.token] = medianPrice;
```

```
277 primaryPrices[cache.token] = medianPrice;
```

CVF-134 INFO

- **Category** Documentation
- **Source** Oracle.sol

Description It is not clear what "stable prices" mean.

Recommendation Consider documenting.

```
283 // to save costs for tokens with stable prices
```

CVF-135 INFO

- **Category** Suboptimal
- **Source** Oracle.sol

Recommendation This line could be simplified using the “applyFactor” function from the “Precision” library.

```
303 price = price * dataStore.getUint(Keys.priceFeedPrecisionKey(token))
    ↪  / Precision.FLOAT_PRECISION;
```

CVF-136 INFO

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

Description Phantom overflow is possible here.

Recommendation Consider using the “muldiv” function.

```
303 price = price * dataStore.getUint(Keys.priceFeedPrecisionKey(token))
    ↪  / Precision.FLOAT_PRECISION;
```

CVF-137 INFO

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

Description Probably an event should be emitted here.

```
307 primaryPrices[token] = price;
```

CVF-138 INFO

- **Category** Suboptimal
- **Source** MarketUtils.sol

Description Annual rates are inefficient.

Recommendation Consider using per-second rates and converting rates between annual to per-second in UI.

```
39 uint256 public constant MAX_ANNUAL_FUNDING_FACTOR = 1000 * Precision
   ↪ .FLOAT_PRECISION;
```

CVF-139 INFO

- **Category** Documentation
- **Source** MarketUtils.sol

Description The number format of these fields is unclear.

Recommendation Consider documenting.

```
42 uint256 indexTokenPrice;
   uint256 longTokenPrice;
   uint256 shortTokenPrice;
```

CVF-140 INFO

- **Category** Suboptimal
- **Source** MarketUtils.sol

Description These structures are identical in terms of their fields, and differ only in their names.

Recommendation Consider using a single structure that could be named “PoolAmount-Delta”.

```
47 event PoolAmountIncrease(
```

```
53 event PoolAmountDecrease(
```

```
59 event ImpactPoolAmountIncreased(
```

```
65 event ImpactPoolAmountDecreased(
```

CVF-141 INFO

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

Recommendation The type of these fields could be more specific.

```
49 address token,
```

```
55 address token,
```

```
61 address token,
```

```
67 address token,
```

```
85 address collateralToken,
```

```
92 address collateralToken,
```

CVF-142 INFO

- **Category** Procedural
- **Source** MarketUtils.sol

Recommendation Consider indexing two first arguments.

```
47 event PoolAmountIncrease(
       address market,
       address token,
50     uint256 amount
   );

   event PoolAmountDecrease(
       address market,
       address token,
       uint256 amount
   );

   event ImpactPoolAmountIncreased(
60     address market,
       address token,
       uint256 amount
   );

   event ImpactPoolAmountDecreased(
       address market,
       address token,
```

CVF-143 INFO

- **Category** Suboptimal
- **Source** MarketUtils.sol

Recommendation These parameters should be indexed.

```
72 address market,
   bool isLong,
```

```
78 address market,
   bool isLong,
```

```
84 address market,
   address collateralToken,
   bool isLong,
```

```
91 address market,
   address collateralToken,
   bool isLong,
```

CVF-144 INFO

- **Category** Suboptimal
- **Source** MarketUtils.sol

Recommendation It would be more efficient to replace each event declaration with two event declarations: one for long and another for short position.

```
73 bool isLong,
```

```
79 bool isLong,
```

```
86 bool isLong,
```

```
93 bool isLong,
```

CVF-145 INFO

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

Description The number format of these arguments in unclear.

```
104 uint256 longTokenPrice,
    uint256 shortTokenPrice,
    uint256 indexTokenPrice
```

CVF-146 INFO

- **Category** Documentation
- **Source** MarketUtils.sol

Description The number format of the returned value is unclear.

Recommendation Consider documenting.

```
107 ) internal view returns (uint256) {
```

CVF-147 INFO

- **Category** Documentation
- **Source** MarketUtils.sol

Description It's not clear why is it possible and why revert division_by_zero is acceptable.

Recommendation Consider documenting.

```
113 // it may be possible for supply to be zero here
```

CVF-148 INFO

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

Description Phantom overflow is possible here.

Recommendation Consider using the "muldiv" function.

```
114 return poolValue * Precision.WEI_PRECISION / supply;
```

CVF-149 INFO

- **Category** Procedural
- **Source** MarketUtils.sol

Recommendation Scaled division should be extracted into a utility function.

```
114 return poolValue * Precision.WEI_PRECISION / supply;
```

CVF-150 INFO

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

Recommendation The return type could be more specific.

```
121 function getOutputToken(address inputToken, Market.Props memory
    ↪ market) internal pure returns (address) {
```

CVF-151 INFO

- **Category** Readability
- **Source** MarketUtils.sol

Recommendation Should be “else if” for readability.

```
125 if (inputToken == market.shortToken) {
```

```
136 if (token == market.shortToken) {
```

```
139 if (token == market.indexToken) {
```

CVF-152 INFO

- **Category** Readability
- **Source** MarketUtils.sol

Recommendation Should be “else revert” for readability.

```
129 revert("MarketUtils: invalid inputToken");
```

```
143 revert("MarketUtils: invalid token");
```

CVF-153 INFO

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

Recommendation The type of the “token” argument could be more specific.

```
132 function getCachedTokenPrice(address token, Market.Props memory
    ↪ market, MarketPrices memory prices) internal pure returns (
    ↪ uint256) {
```

CVF-154 INFO

- **Category** Suboptimal
- **Source** MarketUtils.sol

Recommendation Consider the usage of custom Error entity.

```
143 revert("MarketUtils: invalid token");
```

CVF-155 INFO

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

Description It is not clear why Secondaty price is mixed with Primary prices, consider adding more explanations in comments.

```
150 oracle.getSecondaryPrice(market.indexToken),
    oracle.getPrimaryPrice(market.longToken),
    oracle.getPrimaryPrice(market.shortToken)
```

CVF-156 INFO

- **Category** Suboptimal
- **Source** MarketUtils.sol

Description Here three external calls are performed to the same contract, which is inefficient.

Recommendation Consider refactoring the oracle API to allow fetching several prices in one call.

```
150 oracle.getSecondaryPrice(market.indexToken),
    oracle.getPrimaryPrice(market.longToken),
    oracle.getPrimaryPrice(market.shortToken)
```

CVF-157 INFO

- **Category** Documentation
- **Source** MarketUtils.sol

Description this will not work for deflationary tokens.

Recommendation Consider documenting.

```
168 uint256 longTokenAmount = getPoolAmount(dataStore, market.
       ↪ marketToken, market.longToken);
    uint256 shortTokenAmount = getPoolAmount(dataStore, market.
       ↪ marketToken, market.shortToken);
```

CVF-158 INFO

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

Description This line should be executed only when “openInterest’ is not zero.

```
193 uint256 openInterestInTokens = getOpenInterestInTokens(dataStore,
       ↪ market, isLong);
```

CVF-159 INFO

- **Category** Suboptimal
- **Source** MarketUtils.sol

Recommendation This check wouldn't be necessary if the "decrementUint" function would be used instead of "setUint".

```
222 if (poolAmount < amount) {
```

CVF-160 INFO

- **Category** Suboptimal
- **Source** MarketUtils.sol

Recommendation These three conditional executions, that use the same condition, could be merged into a single conditional statement.

```
333 address reserveToken = isLong ? market.longToken : market.shortToken
    ↪ ;
```

```
335 uint256 reserveTokenPrice = isLong ? prices.longTokenPrice : prices.
    ↪ shortTokenPrice;
```

```
342 if (isLong) {
```

CVF-161 INFO

- **Category** Suboptimal
- **Source** MarketUtils.sol

Recommendation 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.

```
382 decreaseImpactPoolAmount(dataStore, market, token, impactAmount);
```

CVF-162 INFO

- **Category** Suboptimal
- **Source** MarketUtils.sol

Recommendation It would be more efficient to de: longOpenInterest | shortOpenInterest == 0

```
505 if (longOpenInterest + shortOpenInterest == 0) {
```

CVF-163 INFO

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

Description Phantom overflow is possible here.

Recommendation Consider using the “muldiv” function.

```
509 int256 adjustedFactor = (fundingFactor * diffUsd / (longOpenInterest
    ↪  + shortOpenInterest) * durationInSeconds).toInt256();
```

CVF-164 INFO

- **Category** Suboptimal
- **Source** MarketUtils.sol

Description Multiplication after division could lead to precision degradation.

Recommendation Consider performing division at the very end.

```
509 int256 adjustedFactor = (fundingFactor * diffUsd / (longOpenInterest
    ↪  + shortOpenInterest) * durationInSeconds).toInt256();
```

CVF-165 INFO

- **Category** Suboptimal
- **Source** MarketUtils.sol

Recommendation There two conditional calculations could be merged into a single conditional statement.

```
539 uint256 poolAmount = getPoolAmount(dataStore, market.marketToken,
      ↪ isLong ? market.longToken : market.shortToken);
540 uint256 poolTokenPrice = isLong ? prices.longTokenPrice : prices.
      ↪ shortTokenPrice;
```

CVF-166 INFO

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

Description Phantom overflow is possible here.

Recommendation Consider using the “muldiv” function.

```
542 uint256 adjustedFactor = borrowingFactor * openInterest / poolUsd;
```

CVF-167 INFO

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

Description Phantom overflow is possible here.

Recommendation Consider using the “muldiv” function.

```
557 int256 factor = adjustedFactor * multiplier.toInt256() / divisor.
      ↪ toInt256();
```

CVF-168 INFO

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

Recommendation The value 365 days" should be a named constant.

```
558 int256 maxFactor = (MAX_ANNUAL_FUNDING_FACTOR * durationInSeconds /
      ↪ (365 days)).toInt256();
```

CVF-169 INFO

- **Category** Readability
- **Source** MarketUtils.sol

Recommendation Should be "else return" for readability.

```
564 return factor;
```

CVF-170 INFO

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

Description Phantom overflow is possible here.

Recommendation Consider using the "muldiv" function.

```
595 return usdValue * supply / poolValue;
```

```
607 return marketTokenAmount * poolValue / supply;
```

CVF-171 INFO

- **Category** Procedural
- **Source** MarketStore.sol

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

```
16 constructor(RoleStore _roleStore) RoleModule(_roleStore) {}
```

CVF-172 INFO

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

Description 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”.

Recommendation Consider also the name “store”.

```
18 function set(address marketToken, Market.Props memory market)
     ↪ external onlyController {
```

CVF-173 INFO

- **Category** Suboptimal
- **Source** MarketStore.sol

Recommendation Consider emitting an event.

```
20 marketTokens.add(marketToken);
```

```
25 marketTokens.remove(marketToken);
```

CVF-174 INFO

- **Category** Procedural
- **Source** GasUtils.sol

Recommendation The “keeper” parameters should be indexed.

```
21 event KeeperExecutionFee(address keeper, uint256 amount);
   event UserRefundFee(address keeper, uint256 amount, bool success);
```

CVF-175 INFO

- **Category** Procedural
- **Source** GasUtils.sol

Description Use of the “transfer” function is discouraged.

Recommendation Consider using the “call” function instead.

```
45 payable(keeper).transfer(executionFeeForKeeper);
```

CVF-176 INFO

- **Category** Readability
- **Source** GasUtils.sol

Recommendation Should be “else return” for readability.

```
88 return dataStore.getUint(Keys.depositGasLimitKey(false));
```

```
96 return dataStore.getUint(Keys.withdrawalGasLimitKey(false));
```

CVF-177 INFO

- **Category** Documentation
- **Source** GasUtils.sol

Description What if the parameter is not set?

Recommendation Consider documenting this case and maybe add additional check.

```
88 return dataStore.getUint(Keys.depositGasLimitKey(false));
```

```
96 return dataStore.getUint(Keys.withdrawalGasLimitKey(false));
```

CVF-178 INFO

- **Category** Readability
- **Source** GasUtils.sol

Recommendation Should be “else if” for readability.

```
104 if (OrderUtils.isDecreaseOrder(order.orderType())) {
```

```
108 if (OrderUtils.isSwapOrder(order.orderType())) {
```

CVF-179 INFO

- **Category** Readability
- **Source** GasUtils.sol

Recommendation This line should be in an “else” branch for readability.

```
112 OrderUtils.revertUnsupportedOrderType();
```

CVF-180 INFO

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

Recommendation The type of this field should be “IWETH”.

```
35 address weth;
```

CVF-181 INFO

- **Category** Suboptimal
- **Source** WithdrawalUtils.sol

Description String error messages are inefficient.

Recommendation Consider using named errors instead.

```
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");
```

CVF-182 INFO

- **Category** Suboptimal
- **Source** WithdrawalUtils.sol

Description There is no empty market check.

Recommendation Consider adding.

```
78 Market.Props memory market = params.marketStore.get(params.market);
```

```
112 Market.Props memory market = params.marketStore.get(withdrawal.
   ↪ market);
```

CVF-183 INFO

- **Category** Suboptimal
- **Source** WithdrawalUtils.sol

Recommendation Use named args to avoid argument misplacing.

```
80  Withdrawal.Props memory withdrawal = Withdrawal.Props(
        params.account,
        market.marketToken,
        params.marketTokensLongAmount,
        params.marketTokensShortAmount,
        params.minLongTokenAmount,
        params.minShortTokenAmount,
        block.number,
        params.hasCollateralInETH,
        params.executionFee,
90      new bytes32[](0)
    );
```

```
118 cache.poolValue = MarketUtils.getPoolValue(
        params.dataStore,
120     market,
        longTokenPrice,
        shortTokenPrice,
        params.oracle.getPrimaryPrice(market.indexToken)
    );
```

```
131     SwapPricingUtils.GetSwapPricingParams(
            params.dataStore,
            market.marketToken,
            market.longToken,
            market.shortToken,
            longTokenPrice,
            shortTokenPrice,
            -(cache.marketTokensLongUsd.toInt256()),
            -(cache.marketTokensShortUsd.toInt256())
140     )
```

```
    _ExecuteWithdrawalParams memory _params =
        ↪ _ExecuteWithdrawalParams(
        market,
        withdrawal.account,
        market.shortToken,
        market.longToken,
        shortTokenPrice,
        longTokenPrice,
        withdrawal.marketTokensLongAmount,
        withdrawal.hasCollateralInETH,
        cache.marketTokensLongUsd,
        usdAdjustment * cache.marketTokensLongUsd.toInt256() / (
            ↪ cache.marketTokensLongUsd + cache.marketTokensShortUsd
            ↪ ).toInt256()
    );
```

```
    _ExecuteWithdrawalParams memory _params =
        ↪ _ExecuteWithdrawalParams(
        market,
        withdrawal.account,
        market.longToken,
        market.shortToken,
        longTokenPrice,
        shortTokenPrice,
        withdrawal.marketTokensShortAmount,
        withdrawal.hasCollateralInETH,
        cache.marketTokensShortUsd,
        usdAdjustment * cache.marketTokensShortUsd.toInt256() / (
            ↪ cache.marketTokensLongUsd + cache.marketTokensShortUsd
            ↪ ).toInt256()
```

```
GasUtils.payExecutionFee(
    params.dataStore,
    params.withdrawalStore,
    withdrawal.executionFee,
    params.startingGas,
    params.keeper,
    withdrawal.account
```

CVF-184 INFO

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

Recommendation 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.

```
97 bytes32 key = keccak256(abi.encodePacked(nonce));
```

CVF-185 INFO

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

Description Phantom overflow is possible here.

Recommendation Consider using the "muldiv" function.

```
154 usdAdjustment * cache.marketTokensLongUsd.toInt256() / (cache.
    ↪ marketTokensLongUsd + cache.marketTokensShortUsd).toInt256()
```

```
175 usdAdjustment * cache.marketTokensShortUsd.toInt256() / (cache.
    ↪ marketTokensLongUsd + cache.marketTokensShortUsd).toInt256()
```

CVF-186 INFO

- **Category** Suboptimal
- **Source** WithdrawalUtils.sol

Description Transferring fees on each transaction is inefficient.

Recommendation Consider accumulating fees in the pool and allowing the fee receiver to kate them later.

```
231 PricingUtils.transferFees(
```

CVF-187 INFO

- **Category** Suboptimal
- **Source** EnumerableValues.sol

Recommendation Use unchecked declaration here to save gas because you know for sure that always i < type(uint256).max and i >= start

```
17 for (uint256 i = start; i < end; i++) {
       items[i - start] = set.at(i);
   }
```

```
30 for (uint256 i = start; i < end; i++) {
       items[i - start] = set.at(i);
   }
```

```
43 for (uint256 i = start; i < end; i++) {
       items[i - start] = set.at(i);
   }
```

CVF-188 INFO

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

Recommendation The type of these fields could be more specific.

```
19 address tokenA;
20 address tokenB;
```

CVF-189 INFO

- **Category** Documentation
- **Source** SwapPricingUtils.sol

Description The semantics of these fields is unclear.

Recommendation Consider documenting.

```
23 int256 usdDeltaForTokenA;
   int256 usdDeltaForTokenB;
```

```
28 uint256 poolUsdForTokenA;
   uint256 poolUsdForTokenB;
30 uint256 nextPoolUsdForTokenA;
   uint256 nextPoolUsdForTokenB;
```

CVF-190 INFO

- **Category** Suboptimal
- **Source** SwapPricingUtils.sol

Recommendation Unclear logic, consider adding more explanations.

```
function getUsdAdjustment(DataStore dataStore, address market,
    ↪ PoolParams memory poolParams) internal view returns (int256) {
    uint256 initialDiffUsd = Calc.diff(poolParams.poolUsdForTokenA,
        ↪ poolParams.poolUsdForTokenB);
    uint256 nextDiffUsd = Calc.diff(poolParams.nextPoolUsdForTokenA,
        ↪  poolParams.nextPoolUsdForTokenB);
    // check whether an improvement in balance comes from causing
        ↪ the balance to switch sides
    // for example, if there is $2000 of ETH and $1000 of USDC in
        ↪ the pool
    // adding $1999 USDC into the pool will reduce absolute balance
        ↪ from $1000 to $999 but it does not
    // help rebalance the pool much, the isSameSideRebalance value
        ↪ helps avoid gaming using this case
    bool isSameSideRebalance = poolParams.poolUsdForTokenA <=
        ↪ poolParams.poolUsdForTokenB == poolParams.
        ↪ nextPoolUsdForTokenA <= poolParams.nextPoolUsdForTokenB;
    uint256 impactExponentFactor = dataStore.getUint(Keys.
        ↪ swapImpactExponentFactorKey(market));
    if (isSameSideRebalance) {
        bool hasPositiveImpact = nextDiffUsd < initialDiffUsd;
        uint256 impactFactor = dataStore.getUint(Keys.
            ↪ swapImpactFactorKey(market, hasPositiveImpact));
        return PricingUtils.getUsdAdjustmentForSameSideRebalance(
            initialDiffUsd,
            nextDiffUsd,
            hasPositiveImpact,
            impactFactor,
            impactExponentFactor
        );
    } else {
        uint256 positiveImpactFactor = dataStore.getUint(Keys.
            ↪ swapImpactFactorKey(market, true));
        uint256 negativeImpactFactor = dataStore.getUint(Keys.
            ↪ swapImpactFactorKey(market, false));
        return PricingUtils.getUsdAdjustmentForCrossoverRebalance(
            initialDiffUsd,
            nextDiffUsd,
            positiveImpactFactor,
            negativeImpactFactor,
            impactExponentFactor
        );
    }
}
```

CVF-191 INFO

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

Recommendation The type of these fields could be more specific.

```
19 address longToken;
20 address shortToken;
```

CVF-192 INFO

- **Category** Suboptimal
- **Source** PositionPricingUtils.sol

Recommendation This function is never used, consider removing.

```
115 function transferPositionFees(
        FeeReceiver feeReceiver,
        MarketToken marketToken,
        Position.Props memory position,
        bytes32 feeType,
120     PositionFees memory fees
    ) internal returns (PositionFees memory) {
        if (fees.feeReceiverAmount > 0) {
            marketToken.transferOut(position.collateralToken, fees.
               ↪ feeReceiverAmount, address(feeReceiver));
            feeReceiver.notifyFeeReceived(feeType, position.
               ↪ collateralToken, fees.feeReceiverAmount);
        }

        return fees;
    }
```

CVF-193 INFO

- **Category** Suboptimal
- **Source** PositionPricingUtils.sol

Description The expression “fees.spreadAmount + fees.positionFeeAmount + fees.borrowingFeeAmount” is calculated twice.

Recommendation Consider calculating once and reusing.

```
149 fees.feesForPool = fees.spreadAmount + fees.positionFeeAmount + fees
       ↪ .borrowingFeeAmount - fees.feeReceiverAmount;
150 fees.totalNetCostAmount = fees.fundingFeeAmount - (fees.
       ↪ positionFeeAmount + fees.spreadAmount + fees.
       ↪ borrowingFeeAmount).toInt256();
```

CVF-194 INFO

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

Recommendation The type of this field could be more specific.

```
35 address initialCollateralToken;
```

CVF-195 INFO

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

Recommendation The return array is ignored, consider rewriting the function getMarkets to validateMarkets only.

```
94 MarketUtils.getMarkets(marketStore, params.swapPath);
```

CVF-196 INFO

- **Category** Suboptimal
- **Source** OrderUtils.sol

Description Order key is just a hash of the nonce.

Recommendation Just using the nonce as a key would be more efficient.

```
113 bytes32 key = keccak256(abi.encodePacked(nonce));
```

CVF-197 INFO

- **Category** Suboptimal
- **Source** OrderUtils.sol

Description These two function and identical.

Recommendation Consider merging them into one.

```
178 function isPositionOrder(Order.OrderType orderType) internal pure
    ↪ returns (bool) {
```

```
183 function isIncreaseOrder(Order.OrderType orderType) internal pure
    ↪ returns (bool) {
```

CVF-198 INFO

- **Category** Readability
- **Source** OrderUtils.sol

Recommendation Should be “else if” for readability.

```
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) {
```

CVF-199 INFO

- **Category** Suboptimal
- **Source** OrderUtils.sol

Description The bodies of these conditional statements are very similar.

Recommendation Consider merging them together to avoid code duplication.

213 `if (orderType == Order.OrderType.LimitIncrease) {`

227 `if (orderType == Order.OrderType.LimitDecrease) {`

CVF-200 INFO

- **Category** Procedural
- **Source** OrderUtils.sol

Recommendation This should be done after the conditional statement to avoid code duplication.

218 `oracle.setSecondaryPrice(indexToken, acceptablePrice);`

221 `oracle.setSecondaryPrice(indexToken, acceptablePrice);`

232 `oracle.setSecondaryPrice(indexToken, acceptablePrice);`

235 `oracle.setSecondaryPrice(indexToken, acceptablePrice);`

CVF-201 INFO

- **Category** Readability
- **Source** OrderUtils.sol

Recommendation Should be “else revert” for readability.

264 `revert("OrderUtils: unsupported order type");`

CVF-202 INFO

- **Category** Readability
- **Source** OrderUtils.sol

Recommendation Should be “else if” for readability.

```
279 if (orderType == Order.OrderType.LimitSwap) {
```

CVF-203 INFO

- **Category** Readability
- **Source** OrderUtils.sol

Recommendation Should be in an “else” branch.

```
286 revertUnsupportedOrderType();
```

CVF-204 INFO

- **Category** Readability
- **Source** OrderUtils.sol

Recommendation Should be “else if” for readability.

```
306 if (
```

CVF-205 INFO

- **Category** Readability
- **Source** OrderUtils.sol

Recommendation Should be in an “else” branch.

```
318 revertUnsupportedOrderType();
```

CVF-206 INFO

- **Category** Procedural
- **Source** OrderUtils.sol

Description This function is too simple to be extracted.

Recommendation Consider removing this function.

```
321 function revertUnsupportedOrderType() internal pure {
```

CVF-207 INFO

- **Category** Documentation
- **Source** Timelock.sol

Description 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.

Recommendation Consider removing or explaining its role in a documentation comment.

```
8 contract Timelock {
```

CVF-208 INFO

- **Category** Procedural
- **Source** Timelock.sol

Recommendation This variable should be declared as immutable.

```
9 address public admin;
```

CVF-209 INFO

- **Category** Suboptimal
- **Source** Timelock.sol

Recommendation The admin role cannot be transferred, consider adding transferOwnership function or inherit from Ownable openzeppelin contract.

```
9 address public admin;
```

CVF-210 INFO

- **Category** Procedural
- **Source** Timelock.sol

Description This mapping is never used.

Recommendation Consider removing it.

```
12 mapping (string => bool) public allowedSlowKeys;
```

CVF-211 INFO

- **Category** Suboptimal
- **Source** Timelock.sol

Recommendation The array and the loop are redundant here, as all the keys are hardcoded anyway. Just do six assignments instead.

```
29 string[6] memory allowedKeys = [
```

```
38 for (uint256 i = 0; i < allowedKeys.length; i++) {
```

CVF-212 INFO

- **Category** Suboptimal
- **Source** Timelock.sol

Recommendation 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.

```
43 function fastSetUints(DataStore dataStore, string[] memory prefixes,
   ↪  bytes[] memory data, uint256[] memory values) external
   ↪ onlyAdmin {
```

CVF-213 INFO

- **Category** Suboptimal
- **Source** Timelock.sol

Description String error messages are inefficient.

Recommendation Consider using named errors instead.

```
47     require(allowedFastKeys[prefix], "Timelock: invalid key");
```

```
58 require(allowedFastKeys[prefix], "Timelock: invalid key");
```

CVF-214 INFO

- **Category** Documentation
- **Source** Router.sol

Description The word is misspelled.

Recommendation Cosnider replacing with ”expenditures”

```
10 // users will approve this router for token spenditures
```

CVF-215 INFO

- **Category** Procedural
- **Source** Router.sol

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

```
14 constructor(RoleStore _roleStore) RoleModule(_roleStore) {}
```

CVF-216 INFO

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

Recommendation The type of the “token” argument should be “IERC20”.

```
16 function pluginTransfer(address token, address account, address
     ↪ receiver, uint256 amount) external onlyRouterPlugin {
```

CVF-217 INFO

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

Description The argument name “key” is too generic.

Recommendation Consider renaming to “role”.

```
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) {
```

CVF-218 INFO

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

Description These functions should emit some events.

```
18 function grantRole(address account, bytes32 key) external onlyGov {
```

```
23 function revokeRole(address account, bytes32 key) external onlyGov {
```

CVF-219 INFO

- **Category** Suboptimal
- **Source** RoleStore.sol

Recommendation Consider strictly requiring that Role was not granted before via the check require(roleMembers[key].add(account), ”already granted”)

```
18 function grantRole(address account, bytes32 key) external onlyGov {
       roles.add(key);
20     roleMembers[key].add(account);
   }
```

```
23 function revokeRole(address account, bytes32 key) external onlyGov {
       roleMembers[key].remove(account);
   }
```

CVF-221 INFO

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

Recommendation The type of this argument could be more specific.

```
18 address collateralToken,
```

CVF-222 INFO

- **Category** Suboptimal
- **Source** LiquidationUtils.sol

Recommendation The value “type(int256).min” would be more reasonable here.

```
35 order.setAcceptableUsdAdjustment(-type(int256).max);
```

CVF-223 INFO

- **Category** Suboptimal
- **Source** LiquidationUtils.sol

Description This call is an overkill here, as it is able to handle arbitrary order.

Recommendation Could be replaced with: oracle.setSecondaryPrice (indexToken, oracle.getPrimaryPrice (indexToken));

```
41 OrderUtils.setExactOrderPrice(
       params.oracle,
       params.market.indexToken,
       params.order.orderType(),
       params.order.acceptablePrice(),
       params.order.isLong()
   );
```

CVF-224 INFO

- **Category** Documentation
- **Source** LiquidationUtils.sol

Description What if order can only be processed partialy?

Recommendation Consider documenting this case.

```
63 DecreaseOrderUtils.processOrder(params, true);
```

CVF-225 INFO

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

Recommendation The type of this field could be more specific.

```
88 address initialCollateralToken;
```

CVF-226 INFO

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

Description The name ”Numbers” says nothing.

Recommendation Consider renaming to something meaningful.

92
```
struct Numbers {
```

CVF-227 INFO

- **Category** Documentation
- **Source** Order.sol

Description The semantics of these fields is unclear.

Recommendation Consider documenting.

93
```
uint256 sizeDeltaUsd;
uint256 initialCollateralDeltaAmount;
```

96
```
int256 acceptableUsdAdjustment;
```

CVF-228 INFO

- **Category** Documentation
- **Source** Order.sol

Description The number format of these fields is unclear.

Recommendation Consider documenting.

95
```
uint256 acceptablePrice;
```

97
```
uint256 executionFee;
```

CVF-229 INFO

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

Description The name is too generic.

Recommendation Consider renaming to “OrderProps” or just “Order”.

```
111 struct Props {
```

CVF-230 INFO

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

Recommendation The return type could be more specific.

```
126 function initialCollateralToken(Props memory props) internal pure
    ↪ returns (address) {
```

CVF-231 INFO

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

Recommendation The type of the “_value” argument could be more specific.

```
182 function setInitialCollateralToken(Props memory props, address
    ↪ _value) internal pure {
```

CVF-232 INFO

- **Category** Suboptimal
- **Source** SwapOrderUtils.sol

Description The expression “params.order.swapPath()” is calculated several times.

Recommendation Consider calculating once and reusing.

```
17 address firstMarket = params.order.swapPath()[0];
address lastMarket = params.order.swapPath()[params.order.swapPath()
    ↪ .length - 1];
```

CVF-233 INFO

- **Category** Readability
- **Source** SwapOrderUtils.sol

Recommendation Use named args to avoid misplacing.

```
33 (address tokenOut, uint256 outputAmount) = SwapUtils.swap(SwapUtils.
      ↪ SwapParams(
       params.dataStore,
       params.oracle,
       params.feeReceiver,
       params.order.initialCollateralToken(),
       params.order.initialCollateralDeltaAmount(),
       params.swapPathMarkets,
40     params.order.minOutputAmount(),
       address(0)
   ));
```

CVF-234 INFO

- **Category** Documentation
- **Source** OracleModule.sol

Description 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.

Recommendation Consider explaining more.

```
11 // since re-entrancy could allow functions to be called with prices
```

CVF-235 INFO

- **Category** Documentation
- **Source** IPriceFeed.sol

Description The difference between these two values is unclear.

Recommendation Consider documenting.

```
7  uint80 roundId,
```

```
11 uint80 answeredInRound
```

CVF-236 INFO

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

Description The meaning of the word ”answer” is not clear. Answer to which question is it?

Recommendation Consider renaming to just ”price”.

```
8 int256 answer,
```

CVF-237 INFO

- **Category** Documentation
- **Source** IPriceFeed.sol

Description The semantics of these values is unclear.

Recommendation Consider documenting.

```
9  uint256 startedAt,
10 uint256 updatedAt,
```

CVF-238 INFO

- **Category** Procedural
- **Source** MarketFactory.sol

Recommendation Consider declaring immutable.

```
13 MarketStore public marketStore;
```

CVF-239 INFO

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

Recommendation The type of these arguments could be more specific.

```
20 address indexToken,
   address longToken,
   address shortToken
```

CVF-240 INFO

- **Category** Suboptimal
- **Source** MarketFactory.sol

Recommendation Consider declaring as a contract level constant.

```
25 "GMX_MARKET",
```

CVF-241 INFO

- **Category** Procedural
- **Source** MarketFactory.sol

Description This function does nothing.

Recommendation Consider removing it.

```
47 function addSwapMarket() external {}
```

CVF-242 INFO

- **Category** Procedural
- **Source** MarketFactory.sol

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

```
47 function addSwapMarket() external {}
```

CVF-243 INFO

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

Description The name is too generic.

Recommendation Consider renaming to “MarketProps” or just “Market”.

```
6 struct Props {
```

CVF-244 INFO

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

Recommendation The type of this field should be “MarketToken”.

```
7 address marketToken;
```

CVF-245 INFO

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

Recommendation The type of these fields could be more specific.

```
8 address indexToken;
  address longToken;
10 address shortToken;
```

CVF-246 INFO

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

Description It’s not clear should index token be one of ”longToken” or ”shortToken”, so many tokens in the structure requires some explanations in comments.

```
8 address indexToken;
```

CVF-247 INFO

- **Category** Documentation
- **Source** Market.sol

Description It’s nor clear what potentially could be here.

Recommendation Consider documenting.

```
11 bytes32[] data;
```

CVF-248 INFO

- **Category** Suboptimal
- **Source** Governable.sol

Recommendation This contract should be declared as “abstract”, as it is not supposed to be deployed as is, but rather inherited by other contracts.

```
5 contract Governable {
```

CVF-249 INFO

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

Description Abbreviated public identifiers is a bad practice.

Recommendation Consider using full words.

```
6 address public gov;
```

CVF-250 INFO

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

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

```
8 event SetGov(address prevGov, address nextGov);
```

CVF-251 INFO

- **Category** Suboptimal
- **Source** Governable.sol

Description The “prevGov” parameter is redundant as its value could be derived from the previous events.

```
8 event SetGov(address prevGov, address nextGov);
```

CVF-252 INFO

- **Category** Suboptimal
- **Source** Governable.sol

Description The “role” parameter is redundant as its value is always the same.

Recommendation 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.

```
10 error Unauthorized(address msgSender, string role);
```

CVF-253 INFO

- **Category** Suboptimal
- **Source** Governable.sol

Recommendation Consider adding the requirement for prevGov != _gov to be sure that the value has changed.

```
29 gov = _gov;
```

CVF-254 INFO

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

Description This event is emitted even if nothing actually changed.

```
31 emit SetGov(prevGov, _gov);
```

CVF-255 INFO

- **Category** Documentation
- **Source** Reader.sol

Description The number format of these arguments is unclear.

Recommendation Consider documenting.

```
44 uint256 longTokenPrice,
   uint256 shortTokenPrice,
   uint256 indexTokenPrice
```

CVF-256 INFO

- **Category** Documentation
- **Source** Reader.sol

Description The number format of the returned value is unclear.

Recommendation Consider documenting.

```
47 ) external view returns (uint256) {
```

CVF-257 INFO

- **Category** Suboptimal
- **Source** DecreasePositionUtils.sol

Description These variables are redundant.

Recommendation Just give names to the returned values and use them instead.

```
155 ProcessCollateralValues memory values;
```

```
199 PositionPricingUtils.PositionFees memory fees = processPositionCosts
    ↪ (params, prices, position, remainingCollateralAmount);
```

CVF-258 INFO

- **Category** Suboptimal
- **Source** DecreasePositionUtils.sol

Description Here an argument is used as a local variable. This is a bad practice that makes code harder to read.

Recommendation Consider using a separate local variable instead.

```
162 remainingCollateralAmount -= collateralDeltaAmount;
```

CVF-259 INFO

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

Description The code below looks like it is always executed, while it is only executed when the position it not liquidated underwater.

Recommendation Consider putting the rest of the function into an explicit “else” branch to make code easier to read.

```
189 }
```

CVF-260 INFO

- **Category** Readability
- **Source** DecreasePositionUtils.sol

Description Return statements in the middle of a function make code harder to read.

Recommendation Consider refactoring.

```
268 return emptyFees;
```

CVF-261 INFO

- **Category** Suboptimal
- **Source** DecreasePositionUtils.sol

Description Transferring fees on every position change is suboptimal.

Recommendation Consider accumulating fees in the pool and allowing the fee receiver to take them later.

```
272 PricingUtils.transferFees(
```

CVF-262 INFO

- **Category** Readability
- **Source** Precision.sol

Recommendation This value could be rendered as ‘1e30”.

```
10 uint256 public constant FLOAT_PRECISION = 10 ** 30;
```

CVF-263 INFO

- **Category** Readability
- **Source** Precision.sol

Recommendation This value could be rendered as “1e18”.

```
11 uint256 public constant WEI_PRECISION = 10 ** 18;
```

CVF-264 INFO

- **Category** Readability
- **Source** Precision.sol

Description This denominator is n on-standard. Standard denominators are 1e18 and 1e27.

Recommendation Consider using a standard denominator.

```
10 uint256 public constant FLOAT_PRECISION = 10 ** 30;
```

CVF-265 INFO

- **Category** Readability
- **Source** Precision.sol

Recommendation This value could be rendered as “1e12”.

```
13 uint256 public constant FLOAT_TO_WEI_DIVISOR = 10 ** 12;
```

CVF-266 INFO

- **Category** Suboptimal
- **Source** Precision.sol

Recommendation The value for this constant could be calculated as: FLOAT_PRECISION / WEI_PRECISION

```
13 uint256 public constant FLOAT_TO_WEI_DIVISOR = 10 ** 12;
```

CVF-267 INFO

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

Description The constant name is misleading, as this is actually a fixed-point precision, rather than floating point.

```
10 uint256 public constant FLOAT_PRECISION = 10 ** 30;
```

CVF-268 INFO

- **Category** Suboptimal
- **Source** Precision.sol

Description Applying the “toIn256” function to a constant is waste of gas.

Recommendation Consider using plain conversion.

```
20 return amount.toInt256() * factor / FLOAT_PRECISION.toInt256();
```

CVF-269 INFO

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

Description Converting the amount to “int256” before applying factor makes phantom overflow more likely.

Recommendation Consider doing like this: if (factor >= 0) return applyFactor (amount, uint256 (factor)).toInt256 (); else return -applyFactor (amount, uint256 (-factor)).toInt256 ();

```
20 return amount.toInt256() * factor / FLOAT_PRECISION.toInt256();
```

CVF-270 INFO

- **Category** Suboptimal
- **Source** PricingUtils.sol

Description This pair of operations is equivalent to a plain signed subtraction.

```
39 uint256 deltaDiffUsd = Calc.diff(positiveImpactUsd,
   ↪ negativeImpactUsd);
```

```
41 int256 usdAdjustment = Calc.toSigned(deltaDiffUsd, positiveImpactUsd
   ↪  > negativeImpactUsd);
```

CVF-271 INFO

- **Category** Suboptimal
- **Source** PricingUtils.sol

Recommendation Consider using a signed version of PRB Math to bypass this limitation.

```
51 // `PRBMathUD60x18.pow` doesn't work for `x` less than one
```

CVF-272 INFO

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

Recommendation The type of this argument should be “MarketToken”.

```
69 address marketToken,
```

CVF-273 INFO

- **Category** Suboptimal
- **Source** Calc.sol

Recommendation The “abs” call is redundant here. Just do: a + uint256 (b)

```
17 return a + b.abs();
```

CVF-274 INFO

- **Category** Suboptimal
- **Source** Calc.sol

Description The “abs” call is inefficient here, as “b” is known to be negative.

Recommendation Consider doing: return a - uint256 (-b);

```
20 return a - b.abs();
```

CVF-275 INFO

- **Category** Suboptimal
- **Source** Calc.sol

Recommendation These lines could be simplified by specifying “using SafeCase for uint256;”.

```
24 return a + SafeCast.toInt256(b);
```

```
29     return SafeCast.toInt256(a);
```

```
31     return -SafeCast.toInt256(a);
```

CVF-276 INFO

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

Description The name is too generic.

Recommendation Consider renaming to “WithdrawalProps” or just “Withdrawal”.

```
8 struct Props {
```

CVF-277 INFO

- **Category** Readability
- **Source** Bits.sol

Description These formulas don't make the code easier to read.

Recommendation Consider just using "type(uint16)max', "type(uint320.max", and "type(uint64).max".

```
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);
```

CVF-278 INFO

- **Category** Readability
- **Source** RoleModule.sol

Recommendation These modifiers could be replaced with a single modifier: onlyRole (bytes32 role);

```
23 modifier onlyController() {
30 modifier onlyRouterPlugin() {
35 modifier onlyMarketKeeper() {
40 modifier onlyOrderKeeper() {
45 modifier onlyPricingKeeper() {
50 modifier onlyLiquidationKeeper() {
```

CVF-279 INFO

- **Category** Suboptimal
- **Source** RoleModule.sol

Description String error messages are inefficient.

Recommendation Consider using named error instead.

```
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");
```

CVF-280 INFO

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

Description The name is too generic.

Recommendation Consider renaming to “PositionProps” or just “Position”.

```
6 struct Props {
```

CVF-281 INFO

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

Recommendation The type of this field could be more specific.

```
9 address collateralToken;
```

CVF-282 INFO

- **Category** Documentation
- **Source** NonceUtils.sol

Description The semantics of the returned value is unclear.

Recommendation Consider documenting.

```
13 function incrementNonce(DataStore dataStore) internal returns (
      ↪ uint256) {
```

CVF-283 INFO

- **Category** Procedural
- **Source** MarketToken.sol

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

```
9  constructor(RoleStore _roleStore) ERC20("GMX␣Synthetic␣Market", "GD"
      ↪ ) Bank(_roleStore) {
10 }
```

CVF-284 INFO

- **Category** Suboptimal
- **Source** FeeReceiver.sol

Description It's not efficient to use separate contract with an external call just to emit event.

Recommendation Consider emiting the event inside every contract where it is needed.

CVF-285 INFO

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

Recommendation Events are usually named via nouns, such as "Fee" or "ReceivedFee".

```
6 event FeeReceived(bytes32 key, address token, uint256 amount);
```

CVF-286 INFO

- **Category** Procedural
- **Source** FeeReceiver.sol

Recommendation The first two parameters should be indexed.

```
6 event FeeReceived(bytes32 key, address token, uint256 amount);
```

CVF-287 INFO

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

Recommendation The type of the “token” parameter could be more specific.

```
6 event FeeReceived(bytes32 key, address token, uint256 amount);
```

CVF-288 INFO

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

Recommendation The type of the “token” argument could be more specific.

```
8 function notifyFeeReceived(bytes32 key, address token, uint256
    ↪ amount) external {
```

CVF-289 INFO

- **Category** Procedural
- **Source** WithdrawalHandler.sol

Recommendation These variables should be declared as immutable.

```
24 DataStore public dataStore;
   WithdrawalStore public withdrawalStore;
   MarketStore public marketStore;
   Oracle public oracle;
   FeeReceiver public feeReceiver;
```

CVF-290 INFO

- **Category** Readability
- **Source** WithdrawalHandler.sol

Recommendation Consider using named args.

```
61  WithdrawalUtils.CreateWithdrawalParams memory params =
        ↪ WithdrawalUtils.CreateWithdrawalParams(
        dataStore,
        withdrawalStore,
        marketStore,
        account,
        market,
        marketTokensLongAmount,
        marketTokensShortAmount,
        minLongTokenAmount,
70      minShortTokenAmount,
        hasCollateralInETH,
        executionFee,
        EthUtils.weth(dataStore)
    );
```

```
127 WithdrawalUtils.ExecuteWithdrawalParams memory params =
        ↪ WithdrawalUtils.ExecuteWithdrawalParams(
        dataStore,
        withdrawalStore,
130     marketStore,
        oracle,
        feeReceiver,
        key,
        oracleBlockNumbers,
        keeper,
        startingGas
    );
```

CVF-291 INFO

- **Category** Suboptimal
- **Source** WithdrawalHandler.sol

Recommendation Consider direct comparison with a ORACLE_ERROR.

```
96  if (keccak256(abi.encodePacked(reason)) == Keys.ORACLE_ERROR_KEY) {
```

CVF-292 INFO

- **Category** Procedural
- **Source** OrderHandler.sol

Recommendation These variables should be declared as immutable.

```
28 DataStore public dataStore;
   MarketStore public marketStore;
30 OrderStore public orderStore;
   PositionStore public positionStore;
   Oracle public oracle;
   FeeReceiver public feeReceiver;
```

CVF-293 INFO

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

Description These functions should emit some events.

```
56  function createOrder(
```

```
71  function executeOrder(
```

```
105 function updateOrder(
```

```
130 function cancelOrder(bytes32 key) external {
```

CVF-294 INFO

- **Category** Readability
- **Source** OrderHandler.sol

Recommendation Should be “else if” for readability.

```
196 if (OrderUtils.isDecreaseOrder(params.order.orderType())) {
```

```
201 if (OrderUtils.isSwapOrder(params.order.orderType())) {
```

CVF-295 INFO

- **Category** Readability
- **Source** OrderHandler.sol

Recommendation This line should be in an “else” branch for readability.

```
206 OrderUtils.revertUnsupportedOrderType();
```

CVF-296 INFO

- **Category** Procedural
- **Source** OrderHandler.sol

Recommendation This check should be performed earlier.

```
238 OrderUtils.validateNonEmptyOrder(params.order);
```

CVF-297 INFO

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

Recommendation The key type should be “IERC20”.

```
15 mapping (address => uint256) public tokenBalances;
```

CVF-298 INFO

- **Category** Procedural
- **Source** StrictBank.sol

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

```
17 constructor(RoleStore _roleStore) Bank(_roleStore) {}
```

CVF-299 INFO

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

Recommendation The type of the “token” argument should be “IERC20”.

```
19 function recordTransferIn(address token) external onlyController
   ↪ returns (uint256) {
```

```
23 function _recordTransferIn(address token) internal returns (uint256)
   ↪  {
```

```
31 function _afterTransferOut(address token) internal override {
```

CVF-300 INFO

- **Category** Documentation
- **Source** StrictBank.sol

Description WARNING This mechanic will not work in case of reflactions deflationary tokens.

Recommendation Consider documenting this limitation.

```
23 function _recordTransferIn(address token) internal returns (uint256)
   ↪  {
    uint256 prevBalance = tokenBalances[token];
    uint256 nextBalance = IERC20(token).balanceOf(address(this));
    tokenBalances[token] = nextBalance;

    return nextBalance - prevBalance;
}
```

CVF-301 INFO

- **Category** Suboptimal
- **Source** StrictBank.sol

Recommendation It would be more efficient to just subtract the known outgoing transfer amount from the stored balance. Calling an external contract is expensive.

```
32 tokenBalances[token] = IERC20(token).balanceOf(address(this));
```

CVF-302 INFO

- **Category** Procedural
- **Source** DepositHandler.sol

Recommendation These variables should be declared as immutable.

```
22 DataStore public dataStore;
   DepositStore public depositStore;
   MarketStore public marketStore;
   Oracle public oracle;
   FeeReceiver public feeReceiver;
```

CVF-303 INFO

- **Category** Suboptimal
- **Source** DepositHandler.sol

Description String error messages are suboptimal.

Recommendation Consider using named errors instead.

```
44 require(msg.sender == EthUtils.weth(dataStore), "DepositHandler:␣
   ↪ invalid␣sender");
```

CVF-304 INFO

- **Category** Documentation
- **Source** DepositHandler.sol

Description The semantics of the returned value is unclear.

Recommendation Consider giving a descriptive names to the returned value and/or adding a documentation comment.

```
53 ) external nonReentrant onlyController returns (bytes32) {
```

CVF-305 INFO

- **Category** Procedural
- **Source** DepositHandler.sol

Recommendation The feature key could be precomputed in the constructor and stored in an immutable variable.

```
54 FeatureUtils.validateFeature(dataStore, Keys.createDepositFeatureKey
   ↪ (address(this)));
```

CVF-306 INFO

- **Category** Readability
- **Source** DepositHandler.sol

Recommendation Consider using named arguments to avoid mistakes.

```
56 DepositUtils.CreateDepositParams memory params = DepositUtils.
   ↪ CreateDepositParams(
    dataStore,
    depositStore,
    marketStore,
60  account,
    market,
    minMarketTokens,
    hasCollateralInETH,
    executionFee,
    EthUtils.weth(dataStore)
```

CVF-307 INFO

- **Category** Documentation
- **Source** DepositHandler.sol

Description How to cancelDeposit in case if oracle system is broken and this error is permanent?

Recommendation Consider documenting or implementing canceling mechanism.

```
85 if (keccak256(abi.encodePacked(reason)) == Keys.ORACLE_ERROR_KEY) {
       revert(reason);
   }
```

CVF-308 INFO

- **Category** Suboptimal
- **Source** DepositHandler.sol

Recommendation Consider comparing directly reason with Keys.ORACLE_ERROR without hashing.

```
85 if (keccak256(abi.encodePacked(reason)) == Keys.ORACLE_ERROR_KEY) {
```

CVF-309 INFO

- **Category** Suboptimal
- **Source** DepositHandler.sol

Recommendation This check is not needed if you replace ”public” modifier with ”internal”.

```
107 onlySelf
```

CVF-310 INFO

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

Recommendation The return type should be “IWETH”.

```
9 function weth(DataStore dataStore) internal view returns (address) {
```

CVF-311 INFO

- **Category** Suboptimal
- **Source** EthUtils.sol

Recommendation The safer approach would be to use immutable WETH address rather than dynamically set value inside dataStore.

```
10 return dataStore.getAddress(Keys.WETH);
```

CVF-312 INFO

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

Recommendation The type of this field should be “IWETH”.

```
33 address weth;
```

CVF-313 INFO

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

Recommendation The name is too similar to ”ExecuteDepositParams” consider renaming

```
48 struct _ExecuteDepositParams {
```

CVF-314 INFO

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

Recommendation The type of these fields could be more specific.

```
51 address tokenIn;
   address tokenOut;
```

CVF-315 INFO

- **Category** Documentation
- **Source** DepositUtils.sol

Description The number format of these fields is unclear.

Recommendation Consider documenting.

```
53 uint256 tokenInPrice;
   uint256 tokenOutPrice;
```

CVF-316 INFO

- **Category** Documentation
- **Source** DepositUtils.sol

Recommendation The meaning of this attribute is not clear, consider documenting.

```
56 int256 usdAdjustment;
```

CVF-317 INFO

- **Category** Documentation
- **Source** DepositUtils.sol

Description The semantics of the returned value is unclear.

Recommendation Consider giving a descriptive name to the returned value and/or adding a documentation comment.

```
61 function createDeposit(CreateDepositParams memory params) external
   ↪ returns (bytes32) {
```

CVF-318 INFO

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

Description This function should emit some event.

```
61 function createDeposit(CreateDepositParams memory params) external
   ↪ returns (bytes32) {
```

CVF-319 INFO

- **Category** Suboptimal
- **Source** DepositUtils.sol

Recommendation It's safer to use storage variable weth rather than user input argument which may potentially be manipulated.

```
68 if (market.longToken == params.weth) {
       longTokenAmount -= params.executionFee;
70 } else if (market.shortToken == params.weth) {
       shortTokenAmount -= params.executionFee;
   } else {
       uint256 wethAmount = params.depositStore.recordTransferIn(params
          ↪ .weth);
       require(wethAmount == params.executionFee, "DepositUtils:␣
          ↪ invalid␣wethAmount");
   }
```

CVF-320 INFO

- **Category** Documentation
- **Source** DepositUtils.sol

Recommendation Consider adding a comment that the validity of user-passed executionFee is checked below.

```
68 if (market.longToken == params.weth) {
       longTokenAmount -= params.executionFee;
70 } else if (market.shortToken == params.weth) {
       shortTokenAmount -= params.executionFee;
   } else {
       uint256 wethAmount = params.depositStore.recordTransferIn(params
          ↪ .weth);
       require(wethAmount == params.executionFee, "DepositUtils:␣
          ↪ invalid␣wethAmount");
   }
```

CVF-321 INFO

- **Category** Suboptimal
- **Source** DepositUtils.sol

Recommendation Consider emitting an event ExecutionFeeCharged to be able to track it in history.

```
69 longTokenAmount -= params.executionFee;
```

```
71 shortTokenAmount -= params.executionFee;
```

```
73 uint256 wethAmount = params.depositStore.recordTransferIn(params.
   ↪ weth);
   require(wethAmount == params.executionFee, "DepositUtils: invalid
   ↪ wethAmount");
```

CVF-322 INFO

- **Category** Readability
- **Source** DepositUtils.sol

Recommendation Consider the usage of named arguments to avoid misplacing values.

```
77 Deposit.Props memory deposit = Deposit.Props(
       params.account,
       market.marketToken,
80     longTokenAmount,
       shortTokenAmount,
       params.minMarketTokens,
       block.number,
       params.hasCollateralInETH,
       params.executionFee,
       new bytes32[](0)
   );
```

CVF-323 INFO

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

Description It is not clear why calculating bytes32 hash over uint256 value is needed. It makes human readability worse.

Recommendation Consider the usage of uint256 nonce as a key.

```
93 bytes32 key = keccak256(abi.encodePacked(nonce));
```

CVF-324 INFO

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

Description The function should emit an event.

```
100 function executeDeposit(ExecuteDepositParams memory params) internal
    ↪ {
```

CVF-325 INFO

- **Category** Documentation
- **Source** DepositUtils.sol

Recommendation Consider the usage of more descriptive error name.

```
105 revert(Keys.ORACLE_ERROR);
```

CVF-326 INFO

- **Category** Suboptimal
- **Source** DepositUtils.sol

Recommendation Market here potentially coould be removed, consider adding the check of the market validity.

```
108 Market.Props memory market = params.marketStore.get(deposit.market);
```

CVF-327 INFO

- **Category** Readability
- **Source** DepositUtils.sol

Recommendation Consider using named arguments.

```
119     SwapPricingUtils.GetSwapPricingParams(
120         params.dataStore,
            market.marketToken,
            market.longToken,
            market.shortToken,
            longTokenPrice,
            shortTokenPrice,
            (deposit.longTokenAmount * longTokenPrice).toInt256(),
            (deposit.shortTokenAmount * shortTokenPrice).toInt256()
        )
```

```
141     _ExecuteDepositParams memory _params = _ExecuteDepositParams(
            market,
            deposit.account,
            market.longToken,
            market.shortToken,
            longTokenPrice,
            shortTokenPrice,
            deposit.longTokenAmount,
            usdAdjustment * longTokenUsd.toInt256() / (longTokenUsd +
               ↪ shortTokenUsd).toInt256()
150     );
```

```
158     _ExecuteDepositParams memory _params = _ExecuteDepositParams(
            market,
160         deposit.account,
            market.shortToken,
            market.longToken,
            shortTokenPrice,
            longTokenPrice,
            deposit.shortTokenAmount,
            usdAdjustment * shortTokenUsd.toInt256() / (longTokenUsd +
               ↪ shortTokenUsd).toInt256()
        );
```

```
178 GasUtils.payExecutionFee(
        params.dataStore,
180     params.depositStore,
        deposit.executionFee,
        params.startingGas,
        params.keeper,
        deposit.account
    );
```

```
201     depositStore.transferOut(
            EthUtils.weth(dataStore),
            market.longToken,
            deposit.longTokenAmount,
            deposit.account,
            deposit.hasCollateralInETH
        );
```

```
211     depositStore.transferOut(
            EthUtils.weth(dataStore),
            market.shortToken,
            deposit.shortTokenAmount,
            deposit.account,
            deposit.hasCollateralInETH
        );
```

```
222 GasUtils.payExecutionFee(
        dataStore,
        depositStore,
        deposit.executionFee,
        startingGas,
        keeper,
        deposit.account
    );
```

CVF-328 INFO

- **Category** Procedural
- **Source** DepositUtils.sol

Description The products here were already calculated above.

Recommendation Consider using the already calculated values.

```
126 (deposit.longTokenAmount * longTokenPrice).toInt256(),
    (deposit.shortTokenAmount * shortTokenPrice).toInt256()
```

CVF-329 INFO

- **Category** Documentation
- **Source** DepositUtils.sol

Recommendation The phrase ”need to be separately” handled is not clear.

```
137 // this will not work correctly for tokens with a burn mechanism,
        ↪ those need to be separately handled
```

CVF-330 INFO

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

Description The function should emit an event

```
188 function cancelDeposit(
```

CVF-331 INFO

- **Category** Suboptimal
- **Source** DepositUtils.sol

Recommendation Consider checking the validity of market.

```
199 Market.Props memory market = marketStore.get(deposit.market);
```

CVF-332 INFO

- **Category** Documentation
- **Source** DepositUtils.sol

Description ExecutionFee will not be returned.

Recommendation Consider documenting this behaviour in comments or adding the re-turingn of ExeuctionFee.

```
200 if (deposit.longTokenAmount > 0) {
        depositStore.transferOut(
            EthUtils.weth(dataStore),
            market.longToken,
            deposit.longTokenAmount,
            deposit.account,
            deposit.hasCollateralInETH
        );
    }

210 if (deposit.shortTokenAmount > 0) {
        depositStore.transferOut(
            EthUtils.weth(dataStore),
            market.shortToken,
            deposit.shortTokenAmount,
            deposit.account,
            deposit.hasCollateralInETH
        );
```

CVF-333 INFO

- **Category** Suboptimal
- **Source** DepositUtils.sol

Description The “+=” operator here is confusing, as the “mintAmount” value is guaranteed to be zero here.

Recommendation Consider using “=” instead.

```
289 mintAmount += MarketUtils.usdToMarketTokenAmount(
```

CVF-334 INFO

- **Category** Procedural
- **Source** DepositStore.sol

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

```
15 constructor(RoleStore _roleStore) StrictBank(_roleStore) {}
```

CVF-335 INFO

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

Recommendation 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”.

```
17 function set(bytes32 key, Deposit.Props memory deposit) external
   ↪ onlyController {
```

CVF-336 INFO

- **Category** Suboptimal
- **Source** DepositStore.sol

Recommendation Consider emitting an event.

```
19 depositKeys.add(key);
```

```
24 depositKeys.remove(key);
```

CVF-337 INFO

- **Category** Procedural
- **Source** Keys.sol

Description Public constants don't have much sense in a library.
Recommendation Consider declaring as internal.

```
6  bytes32 public constant WETH = keccak256("WETH");
   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");
   bytes32 public constant EXECUTE_WITHDRAWAL_FEATURE = keccak256("
   ↪ EXECUTE_WITHDRAWAL_FEATURE");

15 bytes32 public constant CREATE_ORDER_FEATURE = keccak256("
   ↪ CREATE_ORDER_FEATURE");
   bytes32 public constant EXECUTE_ORDER_FEATURE = keccak256("
   ↪ EXECUTE_ORDER_FEATURE");
   bytes32 public constant UPDATE_ORDER_FEATURE = keccak256("
   ↪ UPDATE_ORDER_FEATURE");
   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");
   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");
   bytes32 public constant EXECUTION_FEE_MULTIPLIER_FACTOR = keccak256(
      ↪ "EXECUTION_FEE_MULTIPLIER_FACTOR");

43 bytes32 public constant DEPOSIT_GAS_LIMIT = keccak256("
      ↪ DEPOSIT_GAS_LIMIT");
   bytes32 public constant WITHDRAWAL_GAS_LIMIT = keccak256("
      ↪ WITHDRAWAL_GAS_LIMIT");
   bytes32 public constant SINGLE_SWAP_GAS_LIMIT = keccak256("
      ↪ SINGLE_SWAP_GAS_LIMIT");
   bytes32 public constant INCREASE_ORDER_GAS_LIMIT = keccak256("
      ↪ INCREASE_ORDER_GAS_LIMIT");
   bytes32 public constant DECREASE_ORDER_GAS_LIMIT = keccak256("
      ↪ DECREASE_ORDER_GAS_LIMIT");
   bytes32 public constant SWAP_ORDER_GAS_LIMIT = keccak256("
      ↪ SWAP_ORDER_GAS_LIMIT");
   bytes32 public constant CANCELLATION_GAS_LIMIT = keccak256("
      ↪ CANCELLATION_GAS_LIMIT");

51 bytes32 public constant MAX_LEVERAGE = keccak256("MAX_LEVERAGE");
   bytes32 public constant MIN_COLLATERAL_USD = keccak256("
      ↪ MIN_COLLATERAL_USD");
```

```
54 string public constant POSITION_IMPACT_FACTOR = "
      ↪ POSITION_IMPACT_FACTOR";
   string public constant POSITION_IMPACT_EXPONENT_FACTOR = "
      ↪ POSITION_IMPACT_EXPONENT_FACTOR";
   string public constant POSITION_SPREAD_FACTOR = "
      ↪ POSITION_SPREAD_FACTOR";
   string public constant POSITION_FEE_FACTOR = "POSITION_FEE_FACTOR";
   string public constant SWAP_IMPACT_FACTOR = "SWAP_IMPACT_FACTOR";
   string public constant SWAP_IMPACT_EXPONENT_FACTOR = "
      ↪ SWAP_IMPACT_EXPONENT_FACTOR";
60 string public constant SWAP_SPREAD_FACTOR = "SWAP_SPREAD_FACTOR";
   string public constant SWAP_FEE_FACTOR = "SWAP_FEE_FACTOR";
   string public constant ORACLE_PRECISION = "ORACLE_PRECISION";
   string public constant OPEN_INTEREST = "OPEN_INTEREST";
   string public constant OPEN_INTEREST_IN_TOKENS = "
      ↪ OPEN_INTEREST_IN_TOKENS";
   string public constant COLLATERAL_SUM = "COLLATERAL_SUM";
   string public constant POOL_AMOUNT = "POOL_AMOUNT";
   string public constant SWAP_IMPACT_POOL_AMOUNT = "
      ↪ SWAP_IMPACT_POOL_AMOUNT";
   string public constant PRICE_FEED = "PRICE_FEED";
   string public constant PRICE_FEED_PRECISION = "PRICE_FEED_PRECISION"
      ↪ ;
70 string public constant STABLE_PRICE = "STABLE_PRICE";
   string public constant RESERVE_FACTOR = "RESERVE_FACTOR";
   string public constant FUNDING_FACTOR = "FUNDING_FACTOR";
   string public constant CUMULATIVE_FUNDING_FACTOR = "
      ↪ CUMULATIVE_FUNDING_FACTOR";
   string public constant CUMULATIVE_FUNDING_FACTOR_UPDATED_AT = "
      ↪ CUMULATIVE_FUNDING_FACTOR_UPDATED_AT";
   string public constant BORROWING_FACTOR = "BORROWING_FACTOR";
   string public constant CUMULATIVE_BORROWING_FACTOR = "
      ↪ CUMULATIVE_BORROWING_FACTOR";
   string public constant CUMULATIVE_BORROWING_FACTOR_UPDATED_AT = "
      ↪ CUMULATIVE_BORROWING_FACTOR_UPDATED_AT";
   string public constant TOTAL_BORROWING = "TOTAL_BORROWING";
```

```
80 string public constant ORACLE_ERROR = "ORACLE_ERROR";
   bytes32 public constant ORACLE_ERROR_KEY = keccak256(abi.
      ↪ encodePacked(ORACLE_ERROR));
```

```
83 string public constant EMPTY_POSITION_ERROR = "EMPTY_POSITION_ERROR"
      ↪ ;
   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";
   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));
```

CVF-338 INFO

- **Category** Suboptimal
- **Source** Keys.sol

Recommendation Consider using the “ORACLE_ERROR” constant as the hash argument.

```
81 bytes32 public constant ORACLE_ERROR_KEY = keccak256(abi.
       ↪ encodePacked(ORACLE_ERROR));
```

CVF-339 INFO

- **Category** Suboptimal
- **Source** Keys.sol

Recommendation Here, “abi.encodePacked” calls could be replaced with simple conversions to “bytes”.

```
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(
```

CVF-340 INFO

- **Category** Suboptimal
- **Source** Keys.sol

Recommendation Consider using the “EMPTY_POSITION_ERROR” constant as the hash argument.

```
84 bytes32 public constant EMPTY_POSITION_ERROR_KEY = keccak256(abi.
   ↪ encodePacked(EMPTY_POSITION_ERROR));
```

CVF-341 INFO

- **Category** Suboptimal
- **Source** Keys.sol

Recommendation Consider using the “UNACCEPTABLE_USD_ADJUSTMENT_ERROR” constant as the hash argument.

```
87 bytes32 public constant UNACCEPTABLE_USD_ADJUSTMENT_ERROR_KEY =
   ↪ keccak256(abi.encodePacked(UNACCEPTABLE_USD_ADJUSTMENT_ERROR))
   ↪ ;
```

CVF-342 INFO

- **Category** Suboptimal
- **Source** Keys.sol

Recommendation Consider using the “INSUFFICIENT_SWAP_OUTPUT_AMOUNT_ERROR” constant as the hash argument.

```
90 bytes32 public constant INSUFFICIENT_SWAP_OUTPUT_AMOUNT_ERROR_KEY =
   ↪ keccak256(abi.encodePacked(
   ↪ INSUFFICIENT_SWAP_OUTPUT_AMOUNT_ERROR));
```

CVF-343 INFO

- **Category** Procedural
- **Source** Keys.sol

Recommendation Naming inconvenience, consider usage of the same prefix Swap both for the constant name and for the function name.

```
302 function impactPoolAmountKey(address market, address token) internal
   ↪  pure returns (bytes32) {
     return keccak256(abi.encodePacked(
         SWAP_IMPACT_POOL_AMOUNT,
```

CVF-344 INFO

- **Category** Procedural
- **Source** LiquidationHandler.sol

Recommendation These variables should be declared as immutable.

```
25 DataStore public dataStore;
   MarketStore public marketStore;
   PositionStore public positionStore;
   Oracle public oracle;
   FeeReceiver public feeReceiver;
```

CVF-345 INFO

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

Recommendation The type of this argument could be more specific.

```
49 address collateralToken,
```

CVF-346 INFO

- **Category** Suboptimal
- **Source** FeatureUtils.sol

Description This basically allows checking arbitrary keys, not only feature-related.

Recommendation Consider hashing the key here with some unique salt to guarantee that only features could be checked.

```
12 return dataStore.getBool(key);
```

CVF-347 INFO

- **Category** Documentation
- **Source** DataStore.sol

Description The semantics of the keys in this mapping is unclear.

Recommendation Consider documenting.

```
 8  mapping(bytes32 => uint256) public uintValues;
    mapping(bytes32 => int256) public intValues;
10  mapping(bytes32 => address) public addressValues;
    mapping(bytes32 => bytes32) public dataValues;
    mapping(bytes32 => bool) public boolValues;
```

CVF-348 INFO

- **Category** Procedural
- **Source** DataStore.sol

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

```
14  constructor(RoleStore _roleStore) RoleModule(_roleStore) {}
```

CVF-349 INFO

- **Category** Suboptimal
- **Source** DataStore.sol

Recommendation These functions are redundant as corresponding mappings are public.

```
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) {
```

CVF-350 INFO

- **Category** Suboptimal
- **Source** DataStore.sol

Description

Recommendation Consider the usage of default NOT_SET value=0, and increment every stored value by 1, to distinguish between initialized and not initialized values.

```
17 return uintValues[key];
```

CVF-351 INFO

- **Category** Suboptimal
- **Source** DataStore.sol

Description Returning an argument as it doesn't make sense.

Recommendation Consider returning the old value or returning nothing.

```
22 return value;
```

```
43 return value;
```

```
64 return value;
```

```
73 return value;
```

```
82 return value;
```

CVF-352 INFO

- **Category** Suboptimal
- **Source** DataStore.sol

Recommendation Consider emitting and event on important data updates.

```
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;
```

CVF-353 INFO

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

Description The words “increment” and “decrement” are usually associated with increasing and decreasing by one, while these functions increase and decrease by arbitrary value.

Recommendation Consider using words “increase” and “decrease”.

```
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) {
```

CVF-354 INFO

- **Category** Suboptimal
- **Source** DataStore.sol

Recommendation This could be simplified as: return uintValues [key] += value;

```
26 uint256 nextUint = uintValues[key] + value;
   uintValues[key] = nextUint;
   return nextUint;
```

```
47 int256 nextInt = intValues[key] + value;
   intValues[key] = nextInt;
   return nextInt;
```

CVF-355 INFO

- **Category** Suboptimal
- **Source** DataStore.sol

Recommendation This could be simplified as: return uintValues [key] -= value;

```
32 uint256 nextUint = uintValues[key] - value;
   uintValues[key] = nextUint;
   return nextUint;
```

```
53 int256 nextInt = intValues[key] - value;
   intValues[key] = nextInt;
   return nextInt;
```

CVF-356 INFO

- **Category** Procedural
- **Source** Bank.sol

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

```
14 constructor(RoleStore _roleStore) RoleModule(_roleStore) {}
```

```
51 function _afterTransferOut(address /* token */) internal virtual {}
```

CVF-357 INFO

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

Recommendation The type of the “token” argument should be “IERC20”.

```
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 {}
```

CVF-358 INFO

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

Recommendation The type of the “weth” argument should be “IWETH”.

```
21 address weth,
```

CVF-359 INFO

- **Category** Procedural
- **Source** Bank.sol

Description 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.

Recommendation Consider initialization of WETH address once.

```
21 address weth,
```

CVF-360 INFO

- **Category** Documentation
- **Source** Bank.sol

Recommendation 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.

```
25 bool hasCollateralInETH
```

CVF-361 INFO

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

Recommendation The type of the “token” argument should be IWETH”.

```
42 function _transferOutEth(address token, uint256 amount, address
   ↪ receiver) internal {
```

CVF-362 INFO

- **Category** Suboptimal
- **Source** Bank.sol

Description Usage of the “transfer” function is discouraged.

Recommendation Consider using “call” instead.

```
46 payable(receiver).transfer(amount);
```

CVF-363 INFO

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

Description This function should accept the amount and the recipient address as arguments.

```
51 function _afterTransferOut(address /* token */) internal virtual {}
```

CVF-364 INFO

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

Description Despite the name, this library don’t contains any utilities, but only certain constants.

Recommendation Consider renaming.

```
5 library FeeUtils {
```

CVF-365 INFO

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

Description The name is too generic.

Recommendation Consider renaming to “DepositProps” or just “Deposit”.

```
6 struct Props {
```

ABDK
Consulting

About us

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

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

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting