Quantstamp

Blex.Io

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	Perpetual futures market
Timeline	2023-07-10 through 2023-09-05
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Source Code	• blex-dex/contracts #8c96f48
	• Roman Rohleder Senior

Documentation quality	Low
Test quality	Medium
Total Findings	47 Fixed: 12 Acknowledged: 32 Mitigated: 3
High severity findings	3 Fixed: 1 Acknowledged: 2

Auditors	• Auditing Engineer • Jonathan Mevs Auditing Engineer • Zeeshan Meghji Auditing Engineer • Cameron Biniamow Auditing Engineer	Medium severity findings	10 Fixed: 2 Acknowledged: 8
		Low severity findings	8 Fixed: 5 Acknowledged: 2 Mitigated: 1
		Undetermined severe	9

rity findings	Acknowledged: 9
Informational findings	17 Fixed: 4 Acknowledged: 11 Mitigated: 2

Summary of Findings

Blex.io is a perpetual trading platform where users can provide liquidity and buy and sell perpetual futures contracts. Blex allows traders to speculate on the future price movement of an underlying asset, such as ETH or BTC, without owning the asset itself. Traders are provided with the option to use leverage, allowing them to control larger positions with a smaller amount of capital, which can amplify both potential profits and losses. A funding mechanism ensures the contract's price closely tracks the underlying asset's spot price. Traders who hold positions in the contract pay or receive funding based on the difference between the contract's price and the spot price. A liquidation mechanism is also implemented to manage the risk of highly leveraged positions. If a trader's position moves against them to a certain extent, their position may be forcibly closed (liquidated) to prevent further losses. Users also have the option to provide single-sided liquidity, generally USDT, to earn rewards via protocol fees. The provided liquidity is then used to fund the trading aspect of the protocol.

During the audit we found 50 issues, ranging from informaitonal to high severity. Reoccuring issues have been high code complexity and centralization of power. We further found that the test suite can be improved as well as code documentation

and several best practices that may be implemented to improve overall code quality.

Update: The Blex team has either fixed, mitigated, or acknowledged all issues in the report. Most issues (32 of 47) have been acknowledged. Many of these acknowledged issues will be fixed in future iterations, for example, BLX-2, BLX-5, and BLX-7. Other issues have been acknowledged as being part of the intended design, including BLX-8 and BLX-10. Finally, some issues were acknowledged but not fixed due to a lack of a specific exploit scenario, including BLX-41. We also note that the auditing team removed three issues from the initial report as they were false positives.

The Blex team also implemented an additional test suite using Foundry. The new test suite includes standard tests and fuzz tests as well. We cannot fully evaluate the quality of the tests since there are multiple test suites without a unified coverage report. However, we encourage the Blex team to continue improving their tests.

ID	DESCRIPTION	SEVERITY	STATUS
BLX-1	User Can Receive Excessive Funds	High	Fixed
BLX-2	LP May Withdraw Excessive Liquidity	High	Acknowledged
BLX-3	Centralization Risks	High	Acknowledged
BLX-4	Use of Unsafe Cast Operations Leading to Distorted Fees and PnL Values	Medium	Fixed
BLX-5	`VaultRouter` Cannot Support Multiple Markets	Medium	Acknowledged
BLX-6	Position Can Be Increased for a Different Account than	Medium	Acknowledged

ID	DESCRIPTION	SEVERITY	STATUS
	the Associated Order		
BLX-7	Incorrect Chainlink Integration	Medium	Acknowledged
BLX-8	Invalid Future Price May Be Used	Medium	Acknowledged
BLX-9	Initialization Safety Checks Bypassed	Medium	Fixed
BLX-10	`onlyInitOr` - Modifier Bypassable by the `DEFAULT_ADMIN_ROLE` Role	Medium	Acknowledged
BLX-11	Overflow During Token Conversion May Lead to Locked Funds	Informational	Acknowledged
BLX-12	Payout of Rewards is Not Guaranteed	Medium	Acknowledged
BLX-13	Insufficient Contract Pausability	Medium	Acknowledged
BLX-14	Privileged Roles and Ownership	Medium	Acknowledged
BLX-15	Mismatch Between `vault`	Low	Fixed

ID	DESCRIPTION	SEVERITY	STATUS
	Parameter and `coreVault`		
BLX-16	**Checks-Effects-Interactions Violation**	• **Low** ⓘ	Acknowledged
BLX-17	**Missing Input Validation**	• **Low** ⓘ	Mitigated
BLX-18	**Renounceable Privileged Roles**	• **Low** ⓘ	Fixed
BLX-19	**Invalid Prices When the** `Price.chainPriceFeed` **Variable Is Uninitialized**	• **Low** ⓘ	Fixed
BLX-20	`VaultRouter.getGlobalPnl()` **May Unnecessarily Revert Due to Over-/Underflow**	• **Low** ⓘ	Acknowledged
BLX-21	`MarketConfigStruct.sol` **Config Maximum Size Checks Off-by-One**	• **Low** ⓘ	Fixed
BLX-22	**Attacker Can Take Ownership of Implementation Contract**	• **Low** ⓘ	Fixed
BLX-23	`FREEZER_ROLE` **Not Revoked From Previous** `VaultRouter`	• **Informational** ⓘ	Fixed

ID	DESCRIPTION	SEVERITY	STATUS
BLX-24	Unlocked Pragma	Informational	Acknowledged
BLX-25	Upgradability	Informational	Acknowledged
BLX-26	Clone-and-Own	Informational	Acknowledged
BLX-27	`maxUserNetSizeLimit` Can Be Circumvented	Informational	Acknowledged
BLX-28	Critical Roles Not Granted Upon Deployment	Informational	Acknowledged
BLX-29	Risky ERC20 Token Transfer Design	Informational	Acknowledged
BLX-30	`delegatecall()` to Untrusted Contract	Informational	Acknowledged
BLX-31	Gas Inefficient Design	Informational	Mitigated
BLX-32	`VAULT_MGR_ROLE` Role Holder Can Embezzle Rewards	Informational	Acknowledged
BLX-33	Unnecessary Use of `SafeMath` in Solidity 0.8.x	Informational	Fixed
BLX-34	Overflow of Order IDs	Informational	Acknowledged

ID	DESCRIPTION	SEVERITY	STATUS
BLX-35	**Invalid** `MarketConfigStruct.VALID_DECREASE_BIT_POSITION` **Overlaps Into** `DECREASE_NUM_LIMIT_BIT_POSITION`	Informational	Fixed
BLX-36	**Unchecked return values**	Informational	Acknowledged
BLX-37	**Test Code Mixed with Production Code**	Informational	Fixed
BLX-38	**Application Monitoring Can Be Improved by Emitting More Events**	Informational	Mitigated
BLX-39	**Over-Engineered and Highly Coupled Contracts**	Undetermined	Acknowledged
BLX-40	**Inconsistent Use of the** `minRateLimit`	Undetermined	Acknowledged
BLX-41	**Missing Reentrancy Guards**	Undetermined	Acknowledged
BLX-42	**Keepers Are A Single Point Of Failure**	Undetermined	Acknowledged
BLX-43	**Orders Do Not Support Expiry**	Undetermined	Acknowledged

ID	DESCRIPTION	SEVERITY	STATUS
	Time		
BLX-44	`FastPriceFeed` **Relies on Centralized Updater**	• Undetermined	Acknowledged
BLX-45	`PositionStore.set()` **and** `PositionStore.remove()` **Callable by All** `ROLE_CONTROLLER` **Roles**	• Undetermined	Acknowledged
BLX-46	**Incorrect Accounting of Order Numbers**	• Undetermined	Acknowledged
BLX-47	`FastPriceFeed.setPricesWithBits()` **Always Updates All Up to 8 Tokens**	• Undetermined	Acknowledged

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Disclaimer

Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

1. The Blex team initially provided documentation to the audit team through high-level architecture and sequence diagrams. No technical documentation or specification was provided. Additionally, tests were not initially present in the repository under review. After meeting with

the Blex team, the audit team obtained technical documentation which was written in Chinese and auto-translated English. Hence, the documents may contain error and fail to grasp the developers' intention, which could lead to risks. The Blex team also provided tests that produced inconsistent and incomplete test coverage results. Test coverage was only calculated for select contracts; therefore, the audit team could not properly evaluate the quality of the tests.

The lack of proper documentation and tests slowed the audit process significantly and prevented the audit team from verifying that all aspects of the system behave as expected. A more robust test suite would have likely detected many of the smaller issues and mistakes in the codebase. **Improving technical documentation and the test suite is highly recommended.**

2. The Blex codebase is complex in size and design. Nearly all contracts in the codebase interact with one another through a variety of methods. For instance, the `Market` contract utilizes the Diamond Storage Pattern with the `OrderMgr`, `PositionAddMgr`, and `PositionSubMgr` contracts.

   Deployment of the contracts includes significant off-chain work, such as making calls directly to the contracts to configure state variables and grant privileged roles. Since most of the deployment process is handled manually, the audit team cannot verify that the contracts are deployed and initialized correctly. Misconfiguration of any given contract in the codebase could lead to unpredictable behavior.
3. Numerous critical subsystems of the Blex platform are centralized and vulnerable to exploits from compromised or malicious privileged roles. A compromised privileged role can often bypass the designed workflows and call lower-level contract functions directly. In many cases, an address granted a privileged role, especially the `DEFAULT_ADMIN_ROLE`, can access functions that directly affect user funds or alter the accounting within a contract. This design approach should be reconsidered to mitigate the risk of stolen funds and broken contracts.
4. Several contracts in the codebase include segments of code that imply the contracts are still in the development and testing phase. For example, the contract `CoreVault.sol` still calls `console.log()` multiple times in the function `_withdraw()`. Further, numerous contracts have commented out code or remaining `TODO`s.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow

- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

Files Included

- `contracts/oracle/FastPriceFeed.sol`
- `contracts/market/MarketValid.sol`
- `contracts/market/PositionSubMgr.sol`
- `contracts/position/PositionBook.sol`
- `contracts/market/MarketRouter.sol`
- `contracts/market/PositionAddMgr.sol`
- `contracts/fee/FundFee.sol`
- `contracts/market/Market.sol`

- contracts/market/OrderMgr.sol
- contracts/market/MarketReader.sol
- contracts/fee/FeeRouter.sol
- contracts/market/MarketConfigStruct.sol
- contracts/vault/VaultReward.sol
- contracts/vault/CoreVault.sol
- contracts/referral/Referral.sol
- contracts/oracle/Price.sol
- contracts/market/MarketFactory.sol
- contracts/order/OrderBook.sol
- contracts/vault/VaultRouter.sol
- contracts/market/GlobalValid.sol
- contracts/order/OrderStore.sol
- contracts/market/MarketLib.sol
- contracts/market/MarketDataTypes.sol
- contracts/order/OrderStruct.sol
- contracts/fee/FeeVault.sol
- contracts/position/PositionStore.sol
- contracts/utils/TransferHelper.sol
- contracts/oracle/ChainPriceFeed.sol
- contracts/vault/RewardDistributor.sol
- contracts/position/PositionStruct.sol
- contracts/utils/EnumerableValues.sol
- contracts/ac/Ac.sol
- contracts/market/MarketStorage.sol
- contracts/utils/Calc.sol
- contracts/market/GlobalDataTypes.sol
- contracts/order/OrderLib.sol

Files Excluded

Every file besides the explicitly included ones above were out of the scope of this audit.

Findings

BLX-1

User Can Receive Excessive Funds

• High Fixed

Update

Fixed in commit 86d548e4447ff025144ecf9a308533567a490686 by replacing the comparison from `<` to `<=`, as suggested.

File(s) affected: `./contracts/market/PositionSubMgrLib.sol`

Description: `PositionSubMgrLib.calculateTransToUser()` does not return a zero value if `_params.collateralDelta.toInt256() + dPNL - fees == 0`, only if it is `< 0`. In the case where this value equals zero, the user will still get transferred `_params.collateralDelta.toInt256() + dPNL`, which is more than they should have received as that amount should just be covering the fees associated with closing this position.

Recommendation: Modify the line of code `if (_params.collateralDelta.toInt256() + dPNL - fees < 0) return 0;` to `if (_params.collateralDelta.toInt256() + dPNL - fees <= 0) return 0;`, including the `<=` check instead of just `<`.

BLX-2
LP May Withdraw Excessive Liquidity

High | Acknowledged

> **Update**
>
> The Blex team will only deploy one vault for this version. Later versions will have their code modified to support multiple vaults if desired. Thus, the issue will be of no consequence for the current version.
>
> We do recommend updating documentation and code comments to make the intentions for the current version clear.

File(s) affected: `./contracts/vault/VaultReward.sol`, `./contracts/vault/CoreVault.sol`, `./contracts/vault/VaultRouter.sol`, `./contracts/vault/RewardDistributor.sol`

Description: The liquidity provider's share is calculated by using the total AUM (Assets Under Management) of the protocol instead of the AUM of the specific vault. As a result, the changes in the AUM of the protocol relative to the vault's AUM can result in a liquidity provider withdrawing more from a vault than their fair share. This can be seen in the code for calculating the share value below:

```
//From ERC4626.sol
function _convertToAssets(
    uint256 shares,
    Math.Rounding rounding
) internal view virtual returns (uint256 assets) {
    uint256 supply = totalSupply();
```

```
        return
            (supply == 0)
                ? _initialConvertToAssets(shares, rounding)
                : shares.mulDiv(totalAssets(), supply,
    rounding);
    }

    //From CoreVault
    function totalAssets()
        public
        view
        override(ERC4626, IERC4626)
        returns (uint256)
    {
        return vaultRouter.getAUM();
    }
```

The above code demonstrates that the total supply of a specific vault's token is used in the calculation, but the total assets of the whole protocol are used instead of the total assets for one vault. We have documented an exploit whereby this discrepancy can be used by a malicious liquidity provider to withdraw more funds than they should be able to.

Exploit Scenario:

1. Assume no fees for simplicity.
2. Suppose there is one vault `coreVault1` that exists with a corresponding `vaultRewards1`.
3. `lp1` calls `vaultRewards1.buy()` with `100` USDC
    1. This mints `100` `coreVault` tokens to `lp1` since it goes through `_initialConvertToShares`
4. `lp2` calls `vaultRewards1buy()` with `100` USDC
    1. This mints `100` `coreVault` tokens to `lp2`.
    2. `vaultCore1` now holds `200` USDC
    3. The total supply of `vaultCore1` tokens is `200`
5. Suppose a second vault is introduced `coreVault2` with `vaultRewards2`.
6. `lp1` calls `vaultRewards2.buy()` with `100` USDC
    1. This mints `100` `coreVault2` tokens to `lp1` since it goes through `_initialConvertToShares`
    2. `100` USDC in the second vault
    3. `VaultRouter.getAUM()` now returns `300` (`200` from first vault, `100` from second vault)
7. `lp1` calls `vaultRewards1.sell()` redeeming all their `100` `coreVault1` tokens
    1. The formula `shares.mulDiv(totalAssets(), supply, rounding);` is used to calculate the `USDC` received by `lp1`
    2. `shares.mulDiv(totalAssets(), supply, rounding);` ⇒
    3. `shares*totalAssets()/supply` ⇒

4. `100 * 300 / 200 ⇒ 150` since `totalAssets` is the total AUM of the protocol and adds up the USDC balance of all vaults
5. `lp1` receives more than the `100` USDC they deposited, and there is only `50` USDC left for `lp2` in `coreVault1`

Recommendation: When calculating the value of the vault's shares, use the vault's total assets instead of the protocol's total assets.

BLX-3 Centralization Risks

High Acknowledged

Update

Point 2 was fixed by the Blex team in commit `9c656d564ea93ef1b36089d3fa3e11a04900830a`. The remaining points have been acknowledged. Some of these points are an intentional part of the design, and some of them will be mitigated at a later time. We recommend that the Blex team add user-facing documentation to make all privileged roles clear so users are aware of any centralization risks. The Blex team provided a more detailed answer below:

1. `FundFee.addSkipTime()` is used to skip the time when the blockchain service is unavailable.
2. This issue has been fixed, commit hash: `9c656d564ea93ef1b36089d3fa3e11a04900830a`.
3. Only administrators can invoke this; later, we will transfer permissions to the timelock/multisig contract.
4. Keeper's permissions are granted to the auto-order contract. After the code stabilizes, we will remove write permissions from the functions on the auto order to make them accessible to everyone.
5. This function is used to delete certain illegal orders. After the system stabilizes, the `OrderMgr.sysCancelOrder` function will be restricted from being called.
6. After the system stabilizes, we will transfer `DEFAULT_ADMIN_ROLE to address(0)`.

File(s) affected: `./contracts/vault/RewardDistributor.sol`, `./contracts/vault/CoreVault.sol`, `./contracts/fee/FeeRouter.sol`, `./contracts/fee/FeeVault.sol`, `./contracts/market/PositionAddMgr.sol`, `./contracts/fee/FundFee.sol`, `./contracts/oracle/FastPriceFeed.sol`

Description: There are a number of access-controlled functions that threaten the integrity of markets on Blex.

1. `FundFee.addSkipTime()` allows the admin account to arbitrarily skip periods of applying for market funding. This can substantially impact all traders using the platform.
2. If the team deploys `FastPriceFeed` without assigning an address to `chainPriceFeed`, any address with the `PRICE_UPDATE_ROLE` can assign arbitrary prices to tokens. This is due to the conditions found in `FastPriceFeed._setPrice()`, which will just assign a price passed in if there is no price feed assigned. This significantly threatens the integrity of this market.
3. In `FastPriceFeed.setLastUpdatedAt()` any address with the `MANGER_ROLE` can arbitrarily assign a time value when the price was last updated.
4. The first condition of `PositionAddMgr.increasePositionWithOrders()` does not revert if the `UpdatePositionInputs` specified is invalid but `isExec == true`. This allows addresses with the Position Keeper Role, `ROLE_POS_KEEPER`, to arbitrarily delete open orders based on the exploit scenario below. This significantly impacts the integrity of a Market on Blex.
5. Similarly, we notice that addresses with the `ROLE_POS_KEEPER` role can intentionally pass in inputs that will cause calls to `Market.execOrderKey()` to revert, which will force `OrderMgr.sysCancelOrder()` to be invoked and also cancel the order. This is a way that a PositionKeeper could arbitrarily delete any order, not just open orders.
6. Further, all contracts that inherit the `AcUpgradable` contract will grant the `DEFAULT_ADMIN_ROLE` to the deployer and a given address (usually the `MarketFactory` contract). Unlike the other privileged roles, **the `DEFAULT_ADMIN_ROLE` can grant or revoke any role to/from any address**. Therefore, it is extremely important to ensure that any address that is granted the `DEFAULT_ADMIN_ROLE` is controlled by a trusted party. If an address granted the `DEFAULT_ADMIN_ROLE` is compromised or controlled by a malicious actor, all functions behind the `onlyRole`, or any similar privileged role modifier, are executable by the malicious actor. The following list explains how a compromised `DEFAULT_ADMIN_ROLE` could exploit restricted sensitive functions:
   - `RewardDistributor.withdrawToken()` is restricted to the `MANAGER_ROLE`. Therefore the deployer of `RewardDistributor` can grant the `MANAGER_ROLE` to any address, who can then withdraw any token amount from the `RewardDistributor` contract.
   - `FeeRouter.withdraw()` is restricted to the `WITHDRAW_ROLE`. Therefore the deployer of `FeeRouter` can grant the `WITHDRAW_ROLE` to any address, who can then withdraw any token amount from the `FeeVault` contract.
   - `FeeVault.withdraw()` is restricted to the `WITHDRAW_ROLE`. Therefore, the `FeeVault` deployer can grant the `WITHDRAW_ROLE` to any address, which can then withdraw any token amount from the `FeeVault` contract.
   - `CoreVault.transferOutAssets()` is restricted to the `ROLE_CONTROLLER`. Therefore, the `CoreVault` deployer can grant the

`ROLE_CONTROLLER` to any address, which can then withdraw any amount of the vault collateral token from the `CoreVault` contract.

- `FeeVault.updateGlobalFundingRate()` is restricted to the `ROLE_CONTROLLER`. Therefore, the `FeeVault` deployer can grant the `ROLE_CONTROLLER` to any address, which can then update the funding rates and last funding time. This could be particularly detrimental since `updateGlobalFundingRate()` does not perform any checks or input validation, as only the `FundFee` contract is expected to be the caller.

Due to the amount of highly privileged roles and their direct control over user funds, as described above, this issue has been assigned a high severity.

Recommendation: Regarding points 1 to 3: Consider whether these functions are necessary for Blex to work as intended and consider removing them from the code.

Regarding points 4 to 5: Revert for all invalid inputs or redefine the validity of an order being executed.

Regarding point 6: There is not one solution to fix all instances where this issue occurs since the use of privileged roles differs in each contract. Some contracts may require a multi-sig to have the `DEFAULT_ADMIN_ROLE` grant and revoke privileged roles as necessary for ongoing contract maintenance. In other cases, the caller of certain privileged functions is known and is always the same. Therefore, the privileged role granted to the known caller should not be granted to any other address, which could be done by refactoring the function `AccessControl.grantRole()`.

Consider if the deployer needs to be granted the `DEFAULT_ADMIN_ROLE` for each contact. If not, refactor the function `AcUpgradable.initialize()` or renounce the role after the contract is initialized.

BLX-4
Use of Unsafe Cast Operations Leading to Distorted Fees and PnL Values

Medium Fixed

Update

The Blex team fixed the issue in commit `bf17af0926004f29e042ab2fabee64f71bc12c4b` by implementing the recommended fix.

File(s) affected: `./contracts/position/PositionBook.sol`, `./contracts/fee/FundFee.sol`, `./contracts/fee/FeeRouter.sol`

Description: The following functions perform unsafe cast operations that could lead to overflows/underflows if the casted variable reaches `type(int256).max` (`57896044618658097711785492504343953926634992332820282019728792003956564819967`):

1. `PositionBook.getPNL()` .
2. `PositionBook.increasePosition()` .
3. `PositionBook.liquidatePosition()` .
4. `PositionBook._getPositionData()` .
5. `PositionBook.getMarketPNL()` .
6. `FundFee._getFundingFee()` .
7. `FeeRouter.getFees()` (L233, L238, L245 and L253).

1.-5.: May return completely distorted PnL values for a given account.
6.: May return a completely distorted funding fee, if a position size reaches type(int256).max.
7.: May return completely distorted position open, close, liquidation and execution fees for `_sizeDelta` values that exceed `type(int256).max` . However, this would require large `feeAndRates[market][kind]` fee values to be exploitable, which can only be set by privileged roles.

Recommendation: We recommend replacing the unsafe cast operations with their safe counterparts from the `SafeCast` library, which is already imported and used in other places of the same contract.

BLX-5

`VaultRouter` Cannot Support Multiple Markets

Medium Acknowledged

> **Update**
> The Blex team will only deploy one vault for this version. Later versions will have their code modified to support multiple vaults if desired. Thus, the issue will be of no consequence for the current version.

File(s) affected: `./contracts/vault/CoreVault.sol` , `./contracts/vault/VaultRouter.sol`

Description: The apparent relationship between the `VaultRouter` and `CoreVault` contracts is unclear, as many contradicting elements exist. Some functionality supports a one-to-many relationship, where a single `VaultRouter` contract interacts with many `CoreVault` contracts. Other functionality indicates a one-to-one relationship, where the `VaultRouter` contract will only interact with one `CoreVault` contract. Regardless of the intended relationship of the two contracts, the `VaultRouter` contract is forced to support a one-to-one

relationship due to the function `VaultRouter.setMarket()`, which overwrites the `vault` argument with the `coreVault` state variable. Thus, all `Market` contracts are mapped to the same `CoreVault` contract, even if a different `CoreVault` contract address is provided as an argument.

```
function setMarket(
    address market,
    address vault
) external onlyRole(MULTI_SIGN_ROLE) {

    vault = address(coreVault);

    require(markets.add(market));
    marketVaults[market] = ICoreVault(vault);
    vaultMarkets[ICoreVault(vault)] = market;
    vaults.add(address(vault));
    _grantRole(ROLE_CONTROLLER, market);
    emit MarketSetted(market, address(vault));
}
```

Another area of conflict is the function `VaultRouter.getUSDBalance()`, which returns the sum of each registered `CoreVault` contract's collateral token balance. Assuming the `setMarket()` function does not overwrite the `vault` argument, `CoreVault` contracts in the `VaultRouter` should have unique collateral tokens. Therefore, `getUSDBalance()` will add the balances of differing tokens, producing incorrect results.

Exploit Scenario:

Multiple Markets Using the Same Core Vault

Assume there are two `Market` contracts (`market0` and `market1`) that use WETH and BTC as their respective `indexToken`. Each of the two `Market` contracts uses USDT as their `collateralToken`. Consider the case where both `Market` contracts are registered with the same `CoreVault` (`vaultUSDT`) in the `VaultRouter` contract.

1. Call `VaultRouter.setMarket(market0, vaultUSDT)` for the first `Market` contract (`market0`) that uses WETH as the `indexToken`.
    - The `vault` argument is overwritten with the immutable state variable `coreVault`.
    - `VaultRouter.marketVaults[market0] = coreVault`.
    - `VaultRouter.vaultMarkets[coreVault] = market0`.
2. Call `VaultRouter.setMarket(market1, vaultUSDT)` for the second `Market` contract (`market1`) that uses BTC as the `indexToken`.
    - The `vault` argument is overwritten with the immutable state variable `coreVault`.

- `VaultRouter.marketVaults[market1] = coreVault`.
- `VaultRouter.vaultMarkets[coreVault] = market1`.

3. Notice how `vaultMarkets[vault]` is overwritten with `market1`.

Multiple Markets Using Unique Core Vaults

Assume there are two `Market` contracts (`market0` and `market1`) that use WETH and BTC as their respective `indexToken`. The two `Market` contracts each use a *different* `collateralToken`. `market0` uses USDC as the `collateralToken` while `market1` uses USDT. Therefore, two `CoreVault` contracts are needed for each collateral token (`vaultUSDC` and `vaultUSDT`). Consider the case where both `Market` contracts are registered with their respective `CoreVault` in the `VaultRouter` contract.

1. Call `VaultRouter.setMarket(market0, vaultUSDC)` for the first `Market` contract (`market0`) that uses WETH as the `indexToken`.
   - The `vaultUSDC` argument is overwritten with the immutable state variable `coreVault`.
   - `VaultRouter.marketVaults[market0] = coreVault`.
   - `VaultRouter.vaultMarkets[coreVault] = market0`.
2. Call `VaultRouter.setMarket(market1, vaultUSDT)` for the second `Market` contract (`market1`) that uses BTC as the `indexToken`.
   - The `vault` argument is overwritten with the immutable state variable `coreVault`.
   - `VaultRouter.marketVaults[market1] = coreVault`.
   - `VaultRouter.vaultMarkets[coreVault] = market1`.
3. Notice how the `Market` contracts are mapped to `coreVault` instead of the provided `vaultUSDC` and `vaultUSDT` arguments.

Recommendation: Determine the relationship between the `VaultRouter` and `CoreVault` contracts. If a one-to-many relationship is desired (registered `Market` contracts can have differing collateral tokens), numerous functions in the `VaultRouter` contract must be refactored. Specifically, the `getUSDBalance()` function must be refactored, and all instances of the `coreVault` state variable must be removed from the `VaultRouter` contract. Otherwise, if a one-to-one relationship is desired (registered `Market` contracts have the same collateral token), major changes to the `VaultRouter` contract will need to occur.

BLX-6
Position Can Be Increased for a Different Account than the Associated Order

Medium Acknowledged

Update

> The Blex team has implemented the appropriate check in another contract outside the scope of the audit.

File(s) affected: `./contracts/market/Market.sol`, `./contracts/market/PositionAddMgr.sol`, `./contracts/market/PositionSubMgr.sol`

Description: Throughout the execution of an order, there is no verification that the account specified in the order being executed is equivalent to the one specified in the `UpdatePositionInputs` passed to the function. As a result, an order can be executed that results in the increase of a position for an account that is different than the one specified in the order. While it is understood that `execOrderKey()` can only be called by position keepers, mistakes do happen and the contracts should not support this possibility.

If a position were to be updated for an account that differs from what is included on the order, Blex's reputation would be at risk as the address that placed the order will not have the resulting position that comes from it.

Recommendation: When validating inputs to `execOrderKey()` ensure the accounts specified by the order itself and the `UpdatePositionInputs` are the same. Such as, `require(exeOrder.account == _params._account)`.

BLX-7
Incorrect Chainlink Integration

Medium | Acknowledged

> **Update**
> The Blex team has replaced the use of the deprecated `latestAnswer()` and `latestRound()` function calls with calls to `latestRoundData()` in commit e025a21.
>
> However, the `updatedAt` timestamp returned by `latestRoundData()` is still not validated to be a sufficiently new price. The Blex team has acknowledged this and will fix the issue in a future iteration.

File(s) affected: `./contracts/oracle/ChainPriceFeed.sol`, `./contracts/oracle/Price.sol`

Description: `ChainPriceFeed` uses `latestRound()` and `latestAnswer()`, both marked as deprecated functions by Chainlink. Although zero price is checked, this can still result in stale data or unexpected results from the Chainlink oracle. Additionally, instead of allowing the price decimals for a token to be directly assigned by the team, they should be read from an oracle.

Recommendation: Consider using `Chainlink.latestRoundData()` to obtain the current price of an asset. Obtain the decimals from the oracle. Add a check to compare the current block timestamp and the timestamp at which the price data is obtained. Consider reverting the transaction if the price data obtained stale.

BLX-8
Invalid Future Price May Be Used

Medium | Acknowledged

> **Update**
> The Blex team acknowledged the issue. The team indicated that they use `_maxTimeDeviation` to perform a time deviation check between the price updater server and `block.timestamp`. We advise that `_maxTimeDeviation` should be a reasonably small value.

File(s) affected: `./contracts/oracle/FastPriceFeed.sol`

Description: The internal function `_setLastUpdatedValues()` performs a few important checks on the `_timestamp` for which the price is updated. In particular, two checks that attempt to ensure that the `_timestamp` falls within the `_maxTimeDeviation` on either side of the current time. However, one of these checks allows for future `_timestamp` values to be used:

```
require(
    _timestamp < block.timestamp + _maxTimeDeviation,
    "FastPriceFeed: _timestamp exceeds allowed range"
);
```

The check would only fail if `_timestamp` is more than `_maxTimeDeviation` seconds in the future. However, it never makes sense to set a price in the future as it is not possible to reliably predict the price in advance. Thus, any `_timestamp` value in the future should be disallowed. If an off-chain bug in the price updater results in an incorrect price being set for the future, the protocol may use the wrong prices for collateral and debt assets.

As a result, the protocol could fail to liquidate unhealthy positions or prematurely liquidate healthy positions, leaving the protocol in debt.

Recommendation: Replace the `_timestamp < block.timestamp + _maxTimeDeviation` check with `_timestamp <= block.timestamp`.

BLX-9
Initialization Safety Checks Bypassed

Medium Fixed

Update
Fixed in commit e562ffd by removing function `OrderMgr.setContracts()`, as suggested.

File(s) affected: `./contracts/market/OrderMgr.sol`, `./contracts/market/Market.sol`

Description: The `Market`, `OrderMgr`, `PositionAddMgr`, and `PositionSubMgr` contracts utilize the diamond storage pattern.

In the `OrderMgr` contract, the setter function `setContracts()` updates only the addresses of the following state variables:

- `positionBook`, `orderBookLong`, `orderBookShort`, `marketValid`, `positionSubMgr`, `positionAddMgr`, `feeRouter`, `vaultRouter`, `globalValid`, and `orderMgr`.

There are other setter functions throughout the `Market` contract that update these same state variables but perform input validation and additional operations for contract initialization. Additionally, the privileged role for `setContracts()` and the other setter functions differs. This means that `setContracts()` restricts access to the `ROLE_CONTROLLER` while the remaining setter functions restrict access to only the `MARKET_MGR_ROLE`.

1. `Market.setOrderBooks()` updates `orderBookShort` and `orderBookLong` after validating the updated addresses and initializing the new `OrderBook` contracts. More importantly, the `setOrderBooks()` function initializes the new `OrderBook` contracts with the `OrderStore` contracts from the old `OrderBook` contracts. This ensures that user orders remain connected to the `Market` contract even after updating the `OrderBook` contracts. Alternatively, updating these state variables via `setContracts()` would **not** validate or initialize the new `OrderBook` contracts with the current `OrderStore` contracts. Therefore, user orders would no longer have any connection to the `Market` contract, which would cause unexpected behavior and potentially broken contracts.
2. `Market.setPositionBook()` updates `positionBook` after validating the updated `PositionBook` contract uses the same `PositionStore` contracts as the previous `PositionBook` contract. The updated `positionBook` address is updated in the `MarketRouter` contract as well. This ensures that the same `PositionStore` contracts are used, which hold important state regarding user positions. Alternatively, updating the `positionBook` state variable via `setContracts()` would **not** validate the `PositionStore` contract addresses set in the new `PositionBook` contract. Therefore, user

positions would no longer have any connection to the `Market` contract, which would cause unexpected behavior and potentially broken contracts.

3. `Market.setMarketValid()` updates the `marketValid` state variable after setting the config data of the new `MarketValid` with the config data from the old `MarketValid` contract. Alternatively, updating the `marketValid` state variable via `setContracts()` would **not** set the config data for the new `MarketValid` contract.
4. `Market.setPositionMgr()` updates the `positionAddMgr` or `positionSubMgr` state variables after validating the updated contract addresses are not the zero address. Alternatively, updating the `positionAddMgr` and `positionSubMgr` state variables via the `setContracts()` would **not** validate the updated contract addresses.
5. `Market.setOrderMgr()` updates the `orderMgr` state variable after validating the updated contract address is not the zero address. Alternatively, updating the `orderMgr` state variable via the `setContracts()` would **not** validate the updated contract address.

Recommendation: Remove the function `setContracts()` from the `OrderMgr` contract.

BLX-10

`onlyInitOr` -Modifier Bypassable by the `DEFAULT_ADMIN_ROLE` Role

Medium Acknowledged

> **Update**
> The Blex team acknowledged the issue and indicated that this is the intended design. The team mentioned that the feature has been intentionally designed for quick updates and maintenance in the short term.

File(s) affected: `./contracts/ac/AcUpgradable.sol`

Description: The modifier `onlyInitOr` of contract `AcUpgradable.sol` is defined as follows:

```
    modifier onlyInitOr(bytes32 _role) {
        bool isDefaultAdmin = hasRole(DEFAULT_ADMIN_ROLE,
_msgSender());
        if (isDefaultAdmin) {
            if (block.timestamp - initBlock >= 3600 * 24)
                revert("ac time passed");
        } else {
```

```
            _checkRole(_role, _msgSender());
        }
        _;
    }
```

If the caller holds role `_role` they can call the corresponding function without further restrictions. Otherwise, if the caller holds the `DEFAULT_ADMIN_ROLE` role, they can only call the corresponding function if the call happens within the threshold window of `3600 * 24` seconds after deployment of that contract.

However, the `DEFAULT_ADMIN_ROLE` role further can grant arbitrary addresses arbitrary roles by calling `AccessControl.grantRole()`, as it is the default role admin for all roles and no other role admins have been setup.

The `DEFAULT_ADMIN_ROLE` can therefore bypass the `onlyInitOr` modifier by also granting themself the target role:

1. Calling `AccessControl.grantRole(TARGET_ROLE, msg.sender)`.
2. Calling the `onlyInitOr` -modifer protected function.

This means this modifier can be bypassed and called at any time by `DEFAULT_ADMIN_ROLE` as long as no other role admin has been assigned, making this modifier behave like a `onlyRole(_role)` modifier.

Recommendation: Please clarify if the checked roles are not in reverse (`DEFAULT_ADMIN_ROLE` should always be able to call and `_role` only within the time threshold) and adjust accordingly or otherwise consider removing this modifier, as it is bypassable by design.

BLX-11
Overflow During Token Conversion May Lead to Locked Funds

• Informational Acknowledged

Update
The Blex team has acknowledged the issue and will use caution when integrating tokens into the protocol.

File(s) affected: `./contracts/utils/TransferHelper.sol`

Description: Function `TransferHelper.formatCollateral()` takes an input `amount` value, its original decimal digits `collateralTokenDigits`, and returns the decimal-adjusted value for `usdDecimal` decimal digits (18). However, with its current implementation the operation may unnecessarily overflow when performing

the first part of the operation `amount * (10 ** uint256(collateralTokenDigits))`, thereby restricting input amounts.

Overflow will only occur when the amount exceeds `type(uint256).max / (10 ** collateralTokenDigits)`. This will not happen for the vast majority of tokens. However, it could occur for unconventional tokens with extremely high balances.

This means any input amounts that exceed `type(uint256).max - 10 ** uint256(collateralTokenDigits)` (For example for USDC, which has 6 decimals: `115792089237316195423570985008687907853269984665640564039457584007913128639935`) would lead to an unnecessary revert during operation, thereby leading to locked funds in the following call-chains where `TransferHelper.formatCollateral()` is called from:

1. `PositionSubMgr._transferToVault()`
    1. `PositionSubMgr._decreaseTransaction()`
        1. `PositionSubMgr._decreasePosition()`
            1. `PositionSubMgr.decreasePosition() external`
            2. `PositionSubMgr._liquidatePosition()`
            3. `PositionSubMgr.liquidatePosition() external`
            4. `PositionSubMgr.decreasePositionFromOrder()`
            5. `PositionSubMgr.execOrderKey() external`
2. `PositionSubMgr._decreaseTransaction()` : See above trace.
3. `PositionSubMgr._decreasePosition()` : See above trace.
4. `PositionSubMgr._approveToFeeVault()` : Dead function, never called.
5. `MarketLib.feeWithdraw()` : Internal function, never called.
6. `MarketLib.vaultWithdraw()`
    1. `PositionSubMgr._decreaseTransaction()` : See above trace.
7. `MarketRouter.calculateEquivalentCollateralAmount()`
    1. `MarketRouter.increasePosition() external`
    2. `MarketRouter._updateOrder()`
        1. `MarketRouter.updateOrder() external`
        2. `MarketRouter._updateOrderFromPosition()`
            1. `MarketRouter.decreasePosition() external`
            2. `MarketRouter.increasePosition() external`
8. `OrderMgr._cancelOrder()`
    1. `OrderMgr.cancelOrderList() external`
    2. `OrderMgr.sysCancelOrder() external`
9. `PositionAddMgr.commitIncreasePosition()`
    1. `PositionAddMgr._increasePosition()`
        1. `PositionAddMgr.increasePositionWithOrders() public`
            1. `PositionAddMgr._execIncreaseOrderKey()`
            2. `PositionAddMgr.execOrderKey() external`
10. `PositionAddMgr._transationsFees()`
    1. `PositionAddMgr._increasePosition()` : See above trace.
11. `TransferHelper.transferIn()` : Dead, never called.
12. `TransferHelper.transferOut()`

1. `PositionSubMgr._decreaseTransaction()` : See trace above.
2. `OrderMgr.cancelOrder()` : See trace above
3. `OrderMgr.cancelOrderList()` `external`

Recommendation: We recommend caution when integrating unconventional tokens into the protocol.

BLX-12
Payout of Rewards is Not Guaranteed

Medium Acknowledged

> **Update**
> The Blex team acknowledged the issue and provided the following explanation:
>
> > When the reward distributor has insufficient rewards, users will not receive any rewards. The reason for doing this is to prevent potential vulnerabilities in the reward contract and minimize losses. We do not intend to modify the code and believe that this approach is aimed at reducing the risk to LP (Liquidity Provider).
>
> We still recommend that the team update the public-facing documentation to inform users about this design decision.

File(s) affected: `./contracts/vault/RewardDistributor.sol`

Description: Contract `RewardDistributor.sol` can only payout as many rewards as it is holding:

```
        uint256 balance =
IERC20(rewardToken).balanceOf(address(this));
        if (amount > balance) {
            amount = balance;
        }

        IERC20(rewardToken).safeTransfer(msg.sender,
amount);
```

Should the contract happen to be in a state of insufficient funds of reward tokens, rewards would not be distributed and all accrued rewards until the contract is sufficiently funded again irreversibly lost.

Recommendation: We recommend implementing additional bookkeeping that accounts for underfunded scenarios and stores the "missing" reward amounts. Additionally there should be a function for distributing any such lost but saved funds for the case when the contract is funded again. Further, this scenario of missing rewards should be communicated to users in public-facing documentation.

BLX-13
Insufficient Contract Pausability

Medium Acknowledged

> **Update**
> The Blex team acknowledged the issue and provided the following explanation:
>
> > In `MarketValid` we have already added the "allowopen" and "allowclose" switches to achieve similar functionality.

File(s) affected: `./contracts/market/MarketRouter.sol`

Description: In case of a hack or other security-related incident, sufficient pausability of critical contract functionality related to the transfer of funds is recommended. This can aid in giving the developers time to identify the issue and eventually mitigate it, before more harm is done and then resume said functions again. In this regard, only the following function(s) may be paused, still leaving funds at risk in case of an incident:

1. `MarketRouter.updateOrder()`.
2. `MarketRouter.decreasePosition()`.
3. `MarketRouter.cancelOrderList()`.
4. `MarketRouter.increasePosition()`.

Should a vulnerability be found that can be exploited from any of the above listed functions, the developers would have a reduced chance in quickly stopping the protocol in order to prevent exploitability until a fix is found and pushed.

Recommendation: We recommend to add a pausability modifier (see `whenNotPaused`) at least to the above functions.

BLX-14
Privileged Roles and Ownership

Medium Acknowledged

> **Update**
>
> The Blex team acknowledged the issue but will not modify the current version. They will consider refactoring in future versions to reduce centralization.
>
> We recommend updating the user-facing documentation to make all privileged roles clear for the current version.

File(s) affected: `./contracts/ac/Ac.sol`, `./contracts/ac/AcUpgradable.sol`, `./contracts/vault/RewardDistributor.sol`, `./contracts/position/PositionStore.sol`, `./contracts/position/PositionBook.sol`, `./contracts/order/OrderStore.sol`, `./contracts/order/OrderBook.sol`, `./contracts/oracle/ChainPriceFeed.sol`, `./contracts/oracle/FastPriceFeed.sol`, `./contracts/oracle/Price.sol`, `./contracts/fee/FeeVault.sol`, `./contracts/fee/FundFee.sol`, `./contracts/fee/FeeRouter.sol`, `./contracts/vault/CoreVault.sol`, `./contracts/vault/VaultRouter.sol`, `./contracts/vault/VaultReward.sol`, `./contracts/market/GlobalValid.sol`, `./contracts/market/MarketValid.sol`, `./contracts/market/MarketRouter.sol`, `./contracts/market/Market.sol`, `./contracts/market/MarketFactory.sol`

Description: Certain contracts have state variables, e.g. `owner`, which provide certain addresses with privileged roles. Such roles may pose a risk to end-users.

This project contains several roles on every level of the protocol with direct access to all protocol parameters, allowing privileged roles to accidentally or with malicious intent to:

1. Break the system (denial of service).
2. Permissioned execution (Allow certain trades, but not others).
3. Distorted execution (Distorted oracles or other system parameters) leading to unfavorable trades.
4. Loss of funds (Transfer of some or all funds to arbitrary addresses).

For a detailed list of roles per contract see the following list:

The `AcUpgradable.sol` contract contains the following privileged roles:

1. `DEFAULT_ADMIN_ROLE`, as initialized during the execution of `_initialize()` to `msg.sender`:
    1. Renounce the role (**and thereby prevent any future calls to the following listed functions!**) by calling `revokeRole()` on oneself.

2. Assign a new `DEFAULT_ADMIN_ROLE` address by calling `transferAdmin()`.
3. Grant/Revoke arbitrary roles (`ROLE_CONTROLLER`, `MANAGER_ROLE`, `ROLE_POS_KEEPER`, ...) to/from addresses by caling `grantRole()` / `revokeRole()`.

2. `_owner`, as initialized during the execution of `_initialize()` to `msg.sender`:
   1. Rennounce the role (**and thereby prevent any future calls to the following listed functions!**) by calling `renounceOwnership()`.
   2. Assign a new `_owner` address by calling `transferOwnership()`.
      - **No other actions! This role and the inheritance of `Ownable` is therefore unnecessary (see "Adherence to Best Practices" point 1.).**

The `Ac.sol` contract contains the following privileged roles:

1. `_owner`, as initialized during the execution of `constructor()` to `msg.sender`:
   1. Rennounce the role (**and thereby prevent any future calls to the following listed functions!**) by calling `renounceOwnership()`.
   2. Assign a new `_owner` address by calling `transferOwnership()`.
   3. **No other actions! This role and the inheritance of `Ownable` is therefore unnecessary (see "Adherence to Best Practices" point 1.).**

The `RewardDistributor.sol` contract inherits from contract `AcUpgradable.sol`, therefore contains all its privileged roles and functionailities and additionally:

1. `MANAGER_ROLE`, as initialized/set through `DEFAULT_ADMIN_ROLE` by calling `grantRole()`:
   1. **Withdraw an arbitrary amount of any token from the contract to an arbitrary address by calling `withdrawToken()`.**
2. `VAULT_MGR_ROLE`, as set through `DEFAULT_ADMIN_ROLE` by calling `grantRole()`:
   1. Update last distribution time (`lastDistributionTime`) to the current block timestamp by calling `updateLastDistributionTime()`.
   2. Initialize the last distribution time, change the tokens per interval and update rewards by calling `setTokensPerInterval()`.
3. `rewardTracker`, as initialized during the execution of `initialize()` to parameter `_rewardTracker`:
   1. Transfer any pending reward tokens to oneself and update the last distribution time by calling `distribute()`.

The `PositionStore.sol` contract inherits from contract `AcUpgradable.sol`, therefore contains all its privileged roles and functionailities and additionally:

1. `ROLE_CONTROLLER` (`onlyController`-modifier), as initialized during the constructor execution to `msg.sender`:
   1. Overwrite the global and an accounts position data by calling `set()`.
   2. Overwrite the global position and remove an accounts position by calling `remove()`.
2. `MANAGER_ROLE` (`onlyAdmin`-modifier), as initialized/set through `DEFAULT_ADMIN_ROLE` by calling `grantRole()`:

1. Change the position book contract address (`positionBook`) and grant it the `ROLE_CONTROLLER` role by calling `setPositionBook()` . **Note: Functions `set()` and `remove()` can be called by any address that holds the `ROLE_CONTROLLER` role, which can be granted independently via `grantRole()` , without the need of calling `setPositionBook()` !**

The `PositionBook.sol` contract inherits from contract `Ac.sol` , therefore contains all its privileged roles and functionailities and additionally:

1. `ROLE_CONTROLLER` (`onlyController` -modifier), as initialized during the initializer execution to parameter `marketAddr` :
   1. Increase/Decrease a positions size (and update related fields) by calling `increasePosition()` / `decreasePosition()` .
   2. Decrease the entire collateral of a position by calling `decreaseCollateralFromCancelInvalidOrder()` .
      - by calling `liquidatePosition()` .
2. `MANAGER_ROLE` (`onlyAdmin` -modifier), as set during initializer execution to `msg.sender` :
   1. **This role has no privileged functions it could call in this contract and is therefore obsolete.**

The `OrderStore.sol` contract inherits from contract `Ac.sol` , therefore contains all its privileged roles and functionailities and additionally:

1. `ROLE_CONTROLLER` (`onlyController` -modifier), as initialized during the constructor execution to `msg.sender` :
   1. Overwrite an existing order by calling `set()` .
   2. Add/Remove orders by calling `add()` / `remove()` .
   3. Delete all orders of a given account by calling `delByAccount()` .
   4. Increment the account-specific order ID by calling `generateID()` .

The `OrderBook.sol` contract inherits from contract `Ac.sol` , therefore contains all its privileged roles and functionailities and additionally:

1. `ROLE_CONTROLLER` (`onlyController` -modifier), as initialized during the constructor execution to `msg.sender` :
   1. Create and add new orders by calling `add()` .
   2. Update certain order fields (`price` , `triggerAbove` , take profit and stop loss) by calling `update()` .
   3. Delete all orders of a given account by calling `removeByAccount()` .
   4. Remove a given order (and potentially its corresponding order pair, if it was a close-position) by calling `remove()` .

The `ChainPriceFeed.sol` contract inherits from contract `Ac.sol` , therefore contains all its privileged roles and functionailities and additionally:

1. `MANAGER_ROLE` (`onlyAdmin` -modifier), as initialized during the constructor execution to `msg.sender` :
   1. Change the price sample size (default: `3`) by calling `setSampleSpace()` .
   2. Set/change price feed data (token feed address and decimals) by calling `setPriceFeed()` . **Note: This function is `onlyInitOr` -modifier protected. However, `DEFAULT_ADMIN_ROLE` can bypass it and always call**

it (as can the `MANAGER_ROLE` role holder), even past the defined time threshold. See the dedicated issue for more information.

The `FastPriceFeed.sol` contract inherits from contract `Ac.sol`, therefore contains all its privileged roles and functionailities and additionally:

1. `MANAGER_ROLE` (`onlyInitOr(MANAGER_ROLE)`-modifier), as granted by `DEFAULT_ADMIN_ROLE` via `grantRole()`:
    1. Set the reference price feed by calling `setPriceFeed()`.
    2. Modify several price calculation parameters by calling:
        1. `setMaxTimeDeviation()`.
        2. `setPriceDuration()`.
        3. `setMaxPriceUpdateDelay()`.
        4. `setSpreadBasisPointsIfInactive()`.
        5. `setSpreadBasisPointsIfChainError()`.
        6. `setMinBlockInterval()`.
        7. `setIsSpreadEnabled()`.
        8. `setLastUpdatedAt()`.
        9. `setMaxDeviationBasisPoints()`.
        10. `setMaxCumulativeDeltaDiffs()`.
        11. `setPriceDataInterval()`.
    3. Define the list of token addresses and their precisions by calling `setTokens()`.
2. `PRICE_UPDATE_ROLE` (`onlyUpdater`-modifier), as granted by `DEFAULT_ADMIN_ROLE` via `grantRole()`:
    1. **Arbitrarily update any tokens' price** by calling `setPrices()`.
    2. **Arbitrarily update any tokens' price and execute orders** by calling `setPricesAndExecute()`.
    3. **Arbitrarily update any tokens' price** by calling `setCompactedPrices()`.
    4. **Arbitrarily update the token price of up to the first 8 tokens (with reduced precision of 32bits)** by calling `setPricesWithBits()`.

The `Price.sol` contract inherits from contract `Ac.sol`, therefore contains all its privileged roles and functionailities and additionally:

1. `MANAGER_ROLE` (`onlyInitOr(MANAGER_ROLE)`-modifier), as granted by `DEFAULT_ADMIN_ROLE` via `grantRole()`:
    1. Change the chain, GMX and "fast price" price feeds by calling `setChainPriceFeed()`, `setGmxPriceFeed()` and `setFastPriceFeed()`, respectively.
    2. Enable/Disable the GMX and "fast price" price feeds by calling `setIsGmxPriceEnabled()` and `setFastPriceEnabled()`.
    3. Modify additional price calculation parameters by calling:
        1. `setAdjustment()`.
        2. `setSpreadBasisPoints()`.
        3. `setSpreadThresholdBasisPoints()`.
        4. `setMaxStrictPriceDeviation()`.
        5. `setStableTokens()`.

The `FeeVault.sol` contract inherits from contract `Ac.sol`, therefore contains all its privileged roles and functionailities and additionally:

1. `ROLE_CONTROLLER` (`onlyController` -modifier), as granted by `DEFAULT_ADMIN_ROLE` via `grantRole()`:
    1. **Arbitrarily change the funding rates for long and short positions by calling** `updateGlobalFundingRate()`.
2. `WITHDRAW_ROLE` (`onlyRole(WITHDRAW_ROLE)` -modifier), as granted by `DEFAULT_ADMIN_ROLE` via `grantRole()`:
    1. **Withdraw any tokens from the contract to another address by calling** `withdraw()`.

The `FundFee.sol` contract inherits from contract `Ac.sol`, therefore contains all its privileged roles and functionailities and additionally:

1. `MANAGER_ROLE` (`onlyAdmin` -modifier), as given to `msg.sender` during constructor execution:
    1. Change the minimum fee rate (default: `2083`) by calling `setMinRateLimit()`.
    2. Change the funding rate of the minority side (default: `0`) by calling `setMinorityFRate()`. **Note: This could lead do both (long- and short-) sides to pay each other non-zero amounts.**
    3. Change the funding interval per market (greater than the minimum of `MIN_FUNDING_INTERVAL = 1 hour`) by calling `setFundingInterval()`.
    4. **Add timeframes that will be deducted from all future funding rate computations by calling** `addSkipTime()`.
2. `ROLE_CONTROLLER` (`onlyController` -modifier), as granted by `DEFAULT_ADMIN_ROLE` via `grantRole()`:
    1. Change the cumulative funding rates per market by calling `updateCumulativeFundingRate()`.

The `FeeRouter.sol` contract inherits from contract `Ac.sol`, therefore contains all its privileged roles and functionailities and additionally:

1. `MANAGER_ROLE` (`onlyAdmin` -modifier), as set to `msg.sender` during constructor execution:
    1. Change the fee vault (`feeVault`) contract address by calling `setFeeVault()`.
    2. Change the fund fee (`fundFee`) contract address by calling `setFundFee()`.
2. `MARKET_MGR_ROLE` (`onlyInitOr(MARKET_MGR_ROLE)` -modifier), as granted by `DEFAULT_ADMIN_ROLE` via `grantRole()`:
    1. Arbitrarily change all fees for all markets by calling `setFeeAndRates()`.
3. `WITHDRAW_ROLE` (`onlyRole(WITHDRAW_ROLE)` -modifier), as granted by `DEFAULT_ADMIN_ROLE` via `grantRole()`:
    1. Transfer arbitrary token amounts to some address by calling `withdraw()`.
4. A market address that has the `allowOpen` or `allowClose` flag set (`onlyMarket` -modifier), as defined by the `factory` contract (`getMarkets()`):
    1. Change the cumulative funding rates per market by calling `updateCumulativeFundingRate()`.

5. Following function is callable by a market as described above OR `ROLE_CONTROLLER`, as granted by `DEFAULT_ADMIN_ROLE` via `grantRole()`:
    1. Transfer any pending token allowance from oneself to the fee vault contract (`feeVault`) by calling `collectFees()`.

The `CoreVault.sol` contract inherits from contract `AcUpgradable.sol`, therefore contains all its privileged roles and functionailities and additionally:

1. `MANAGER_ROLE` (`onlyAdmin`-modifier), as initialized/set through `DEFAULT_ADMIN_ROLE` by calling `grantRole()`:
    1. Change the vault router contract address (`vaultRouter`) by calling `setVaultRouter()`.
    2. Change buy/sell transaction costs by calling `setLpFee()`.
    3. Arbitrarily change the withdraw cooldown time (default: `15 minutes`) by calling `setCooldownDuration()`.
2. `ROLE_CONTROLLER` (`onlyController`-modifier), as set to `_vaultRouter` during `initializer()` execution:
    1. **Transfer arbitrary asset amounts out of the vault** by calling `transferOutAssets()`.
3. `FREEZER_ROLE` (`onlyFreezer`-modifier), as set to `_vaultRouter` during `initializer()` execution:
    1. Freeze/unfreeze vault interactions by calling `setIsFreeze()`.

The `VaultRouter.sol` contract inherits from contract `AcUpgradable.sol`, therefore contains all its privileged roles and functionailities and additionally:

1. Addresses that are store in `markets` (`onlyMarket`-modifier), as set via `setMarket()` by `MULTI_SIGN_ROLE`:
    1. Update internal bookkeeping (`fundsUsed[]` and `totalFundsUsed`) regarding borrows and repays while vault is not frozen by calling `borrowFromVault()` / `repayToVault()`.
2. `FREEZER_ROLE` (`onlyFreezer`-modifier), as set via `grantRole()` by `DEFAULT_ADMIN_ROLE`:
    1. Freeze/unfreeze different vault interactions depending on type by calling `setIsFreeze()`.
3. `MULTI_SIGN_ROLE` (`onlyRole(MULTI_SIGN_ROLE)`-modifier), as set via `grantRole()` by `DEFAULT_ADMIN_ROLE`:
    1. Add a market address (and granting it the `ROLE_CONTROLLER` role) by calling `setMarket()`. **Note: While the contract contains logic to allow for multiple vault contracts, function `setMarket()` overwrites the `vault` parameter and only ever uses `coreVault`.**
4. `VAULT_MGR_ROLE` (`onlyRole(VAULT_MGR_ROLE)`-modifier), as set via `grantRole()` by `DEFAULT_ADMIN_ROLE`:
    1. Remove a market address (and revoking its `ROLE_CONTROLLER` role) by calling `removeMarket()`. **Note: While only `MULTI_SIGN_ROLE` can add markets, only `VAULT_MGR_ROLE` can remove markets.**
5. `ROLE_CONTROLLER` (`onlyController`-modifier), as set via `grantRole()` by `DEFAULT_ADMIN_ROLE` or via `setMarket()` by `MULTI_SIGN_ROLE`:

    1. **Transfer arbitrary vault asset amounts to and from any address by calling** `transferToVault()` **/** `transferFromVault()` .

The `VaultReward.sol` contract inherits from contract `AcUpgradable.sol` , therefore contains all its privileged roles and functionailities and additionally:

1. `ROLE_CONTROLLER` (`onlyController` -modifier), as set via `grantRole()` by `DEFAULT_ADMIN_ROLE` :
    1. **Claim LP rewards for any account by calling** `claimForAccount()` .
2. `VAULT_MGR_ROLE` (`onlyRole(VAULT_MGR_ROLE)` -modifier), as set via `grantRole()` by `DEFAULT_ADMIN_ROLE` :
    1. Set the `apr` variable by calling `setAPR()` . **Note: This APR variable remains otherwise unused throughout the codebase.**

The `GlobalValid.sol` contract inherits from contract `Ac.sol` , therefore contains all its privileged roles and functionailities and additionally:

1. `GLOBAL_MGR_ROLE` (`onlyInitOr(GLOBAL_MGR_ROLE)` -modifier), as set via `grantRole()` by `DEFAULT_ADMIN_ROLE` :
    1. Define several parameters for position size limits (in basis points) by calling:
        1. `setMaxSizeLimit()` (default: `10000`).
        2. `setMaxNetSizeLimit()` (default: `10000`).
        3. `setMaxUserNetSizeLimit()` (default: `10000`).
        4. `setMaxMarketSizeLimit()` .

The `MarketValid.sol` contract inherits from contract `Ac.sol` , therefore contains all its privileged roles and functionailities and additionally:

1. `MARKET_MGR_ROLE` (`onlyInitOr(MARKET_MGR_ROLE)` -modifier), as set via `grantRole()` by `DEFAULT_ADMIN_ROLE` :
    1. Indirectly set the market configuration data through `setConf()` .
    2. Directly set the market configuration data through `setConfData()` . **Note: Setting the configuration like this bypasses internal size checks and may use otherwise undefined fields**

The `MarketRouter.sol` contract inherits from contract `AcUpgradable.sol` , therefore contains all its privileged roles and functionailities and additionally:

1. `MARKET_MGR_ROLE` (`onlyInitOr(MARKET_MGR_ROLE)` -modifier), as set via `grantRole()` by `DEFAULT_ADMIN_ROLE` :
    1. Change the position book contract address by calling `updatePositionBook()` . **Note: The callers address must be in the** `markets` **set. However, this can be bypassesd by calling** `addMarket()` **/** `removeMarket()` **.**
    2. Enable/Disable converting positions and orders by calling `setIsEnableMarketConvertToOrder()` .
    3. Add/Remove market addresses by calling `addMarket()` / `removeMarket()` .

The `Market.sol` contract inherits from contract `Ac.sol` , therefore contains all its privileged roles and functionailities and additionally:

1. `ROLE_CONTROLLER` (`onlyController` -modifier), as set via `grantRole()` by `DEFAULT_ADMIN_ROLE` :

    1. Increase/Decrease arbitrary positions by calling `increasePositionWithOrders()` / `decreasePosition()`.
    2. Call any function of the order manager contract (`orderMgr`) in the context of the market storage (`delegatecall`) by calling `fallback()`.
2. `ROLE_POS_KEEPER` (`onlyPositionKeeper` -modifier), as set via `grantRole()` by `DEFAULT_ADMIN_ROLE`:
    1. Liquidate positions by calling `liquidatePositions()`.
    2. Execute orders by calling `execOrderKey()`.
3. `MANAGER_ROLE` (`onlyInitOr(MANAGER_ROLE)` - and `onlyAdmin` - modifiers), as set via `grantRole()` by `DEFAULT_ADMIN_ROLE`:
    1. Add/Remove arbitrary addresses from the plugins by calling `addPlugin()` / `removePlugin()`. **Note: Plugins are hooks into position and order operations and execute code after such operations.**
4. `MARKET_MGR_ROLE` (`onlyRole(MARKET_MGR_ROLE)` -modifier), as set via `grantRole()` by `DEFAULT_ADMIN_ROLE`:
    1. Set the order book, position book, market valid, position manager and order manager contract addresses by calling:
        1. `setOrderBooks()`.
        2. `setPositionBook()`.
        3. `setMarketValid()`.
        4. `setPositionMgr()`.
        5. `setOrderMgr()`. **Note: Any function of this contract could be executed by `ROLE_CONTROLLER` within the context of the market storage (`delegatecall`).**

The `MarketFactory.sol` contract inherits from contract `Ac.sol`, therefore contains all its privileged roles and functionailities and additionally:

1. `MARKET_MGR_ROLE` (`onlyRole(MARKET_MGR_ROLE)` -modifier), as set via `grantRole()` by `DEFAULT_ADMIN_ROLE`:
    1. Remove addresses from the `markets` array by calling `remove()`.
    2. Setup new markets, initialize them and setup roles by calling `create()`.

Having such a complex role system with such powerful abilities, increases the chance of compromised private keys, which in turn could lead to a denial of service and/or loss of funds of the protocol.

Recommendation: This centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows to the owner.

BLX-15
Mismatch Between `vault` Parameter and `coreVault`

• Low Fixed

Update

> The Blex team has fixed the issue by implementing the recommended fix in commit ecb9ff.

File(s) affected: `./contracts/vault/VaultReward.sol`

Description: The `VaultReward.buy()` function accepts a `vault` parameter, which determines the underlying asset the user deposits into `VaultRewards` and is also used for calling the `deposit()` function. However, `VaultReward.buy()` does not approve the `vault` address to transfer tokens. Rather, it approves `coreVault` to transfer these tokens. This implies that the only acceptable parameter for `vault` is `coreVault`. While most users would supply the correct parameter, if a user made the mistake of supplying an incorrect vault or an older vault address, then there is a chance that the transaction may succeed. Then the user's tokens may be transferred to `VaultRewards` without being transferred to `coreVault`. However, `coreVault` would still have pending approval from `VaultRewards`. This might allow an attacker to take advantage of the pending approval to `coreVault`.

Recommendation: Remove the `vault` parameter from the `buy()` function and use the `coreVault` state variable. Also, remove the `vault` parameter from all other relevant functions in `VaultRewards`, including the `sell()` function.

BLX-16
Checks-Effects-Interactions Violation

Low Acknowledged

> **Update**
> The Blex team acknowledged the issue but will not modify the current version. They will consider refactoring in future versions.

File(s) affected: `./contracts/market/MarketRouter.sol`

Description: The Checks-Effects-Interactions coding pattern is meant to mitigate any chance of other contracts manipulating the state of the blockchain in unexpected and possibly malicious ways before control is returned to the original contract. As the name implied, only after checking whether appropriate conditions are met and acting internally on those conditions should any external calls to, or interactions with, other contracts be made. In `MarketRouter.increasePosition()` funds are transferred from the caller to the market before conducting all necessary checks or effects. This is in violation of the common Checks-Effects-Interaction pattern

Recommendation: We recommend refactoring the code so that it conforms to the Checks-Effects-Interactions pattern.

BLX-17
Missing Input Validation

Low Mitigated

> **Update**
> Mitigated in commits 353dc34, 8cce8eb and 8cce8eb by adding most of the recommended checks.

File(s) affected: `./contracts/vault/RewardDistributor.sol`, `./contracts/oracle/FastPriceFeed.sol`, `./contracts/oracle/Price.sol`, `./contracts/fee/FeeVault.sol`, `./contracts/fee/FeeRouter.sol`, `./contracts/vault/CoreVault.sol`, `./contracts/vault/VaultRouter.sol`, `./contracts/market/MarketConfigStruct.sol`

Related Issue(s): SWC-123

Description: It is important to validate inputs, even if they only come from trusted addresses, to avoid human error. The following functions are missing adequate input validation:

1. `AcUpgradable.transferAdmin(): to` not checked to be different from the zero address. Contract may be left without an owner, and functions behind the `onlyOwner` modifier will no longer be executable. (**Update:** Fixed)
2. `AcUpgradable.grantControllerRoleByMarketManager(): _account` not checked to be different from the zero address. (**Update:** Fixed)
3. `RewardDistributor.setTokensPerInterval(): _amount` not checked to be different from zero.
4. `MarketRouter.initialize(): _f` not checked to be different from the zero address. (**Update:** Fixed)
5. `OrderMgr.setContracts(): addrs` elements are not checked to be different from the zero address. (**Update:** Deprecated, as removed due to BLX-11)
6. `FastPriceFeed.setMaxTimeDeviation(): _deviation` not checked to be different from zero.
7. `FastPriceFeed.setMaxPriceUpdateDelay(): _delay` not checked to be different from zero or otherwise reasonably bound.
8. `FastPriceFeed.setSpreadBasisPointsIfInactive(): _point` not checked to be different from zero or otherwise reasonably bound.
9. `FastPriceFeed.setSpreadBasisPointsIfChainError(): _point` not checked to be different from zero or otherwise reasonably bound.
10. `FastPriceFeed.setMinBlockInterval(): _interval` not checked to be different from zero or otherwise reasonably bound.

11. `FastPriceFeed.setMaxDeviationBasisPoints()`: `_maxDeviationBasisPoints` not checked to be different from zero or otherwise reasonably bound.
12. `FastPriceFeed.setMaxCumulativeDeltaDiffs()`: `_maxCumulativeDeltaDiffs[]` not checked to be different from zero or otherwise reasonably bound.
13. `Price.setMaxStrictPriceDeviation()`: `_maxStrictPriceDeviation` not checked to be reasonably bound.
14. `FeeVault.updateGlobalFundingRate()`: `timestamp` not checked to be the current timestamp or otherwise within a reasonable window of tolerance and can therefore be arbitrarily set.
15. `FeeVault.updateGlobalFundingRate()`: `market` not checked to be different from the the zero address.
16. `FeeRouter.setFeeAndRates()`: `rates.length` not checked to be smaller than `FeeType.Counter (12)`. Further, `rates[]` values are not checked to be smaller than `FEE_RATE_PRECISION` for open and close fee types or smaller than `type(int256).max` for all other types. (**Update:** Fixed)
17. `FeeRouter._getFee()`: `kind` not checked to be smaller than `FeeType.Counter (12)`. (**Update:** Deprecated due to the removal of the function)
18. `CoreVault.initialize()`: `_asset`, `_vaultRouter`, `_feeRouter`, and `_vaultReward` are not checked to be different from the zero address. (**Update:** Fixed)
19. `CoreVault.setLpFee()`: `fee` not checked to be non-zero or otherwise reasonably bound.
20. `CoreVault.setVaultRouter()`: `_vaultRouter` not checked to be different from the zero address. (**Update:** Fixed)
21. `CoreVault.setCooldownDuration()`: `_duration` not checked to be non-zero or otherwise reasonably bound.
22. `VaultRouter.setIsFreeze()`: `freezeType` not checked to be smaller than `3`. If an invalid `freezeType` of `3` or higher is given, the call succeeds and the event `LogIsFreeze` is emitted with an invalid `freezeType`. (**Update:** Fixed)
23. `VaultRouter.setMarket()`: `vault` is overwritten with `coreVault` in `L62` and is therefore spurious. (**Update:** Deprecated, due to fix for BLX-6)
24. `MarketConfigStruct.setDecimals()`: `_decimals` not checked to fit into 8 bits, i.e. being smaller than `(2 ** 8) - 1`. (**Update:** Fixed)

Recommendation: We recommend adding the relevant checks.

BLX-18
Renounceable Privileged Roles

Low Fixed

Update

> Fixed in commit d454f7b by overwriting `renounceRole()` in contract `AcUpgradable.sol`, as suggested.

File(s) affected: `all`

Description: If the address granted the `DEFAULT_ADMIN_ROLE` calls `renounceRole(DEFAULT_ADMIN_ROLE)` in any contract in the codebase, and no other addresses have the `DEFAULT_ADMIN_ROLE`, critical functionality guarded by the modifier `onlyRole(DEFAULT_ADMIN_ROLE)` will not be executable. Additionally, if no address is granted the `DEFAULT_ADMIN_ROLE`, many other critical privileged roles cannot be granted.

Recommendation: Confirm that this is the intended behavior. If not, override and disable the `renounceRole()` function in the affected contracts. For extra security, consider using a two-step process when transferring the ownership of the contract (e.g. `Ownable2Step` from OpenZeppelin).

BLX-19
Invalid Prices When the `Price.chainPriceFeed` Variable Is Uninitialized

Low Fixed

> **Update**
> Fixed in commit 9c656d5 by now requiring `chainPriceFeed` to be non-zero, as suggested.

File(s) affected: `./contracts/oracle/Price.sol`

Description: In case `Price.chainPriceFeed` is never set via `setChainPriceFeed()`, function `getPrice()` would return distorted prices, as in that case `getChainPrice()` will return `0`. Depending on whether `FastPrice` or `GmxPrice` and price adjustments are enabled, the result of `getPrice()` for any token may be as low as `ONE_USD` (`PRICE_PRECISION = 10 ** 30`).

Recommendation: Consider adding a check in `_getPrice()` or `getChainPrice()` to revert in case of uninitialized `chainPriceFeed`.

BLX-20

`VaultRouter.getGlobalPnl()` May Unnecessarily Revert Due to Over-/Underflow

Low Acknowledged

> **Update**
> The Blex team has acknowledged the issue and indicated that underflows and overflows should not occur.

File(s) affected: `./contracts/vault/VaultRouter.sol`

Description: Function `VaultRouter.getGlobalPnl()` adds up the profit-and-loss across all markets, storing the result in `int256 pnl` while iterating over `markets.values()`. In unlucky situations, where several markets in a row have very high and all positive or all negative PnL, the iteration may over-/underflow, while the overall result could still have been stored inside an `int256` variable.

Recommendation: Clarify if this could occur in your targeted setup and, if necessary, consider redesigning by sorting and iteratively adding positve and then negative PnL of similar sizes.

BLX-21

`MarketConfigStruct.sol` Config Maximum Size Checks Off-by-One

Low Fixed

> **Update**
> Fixed in commit ed06b18 by changing the upper bound checks as suggested.

File(s) affected: `./contracts/market/MarketConfigStruct.sol`

Description: Contract `MarketConfigStruct.sol` provides functions for setting market-related configuration parameters using a size-efficient bit-field.

In order to check, whether a given input value of type `uint256` fits into the corresponding bit-field at its corresponding bit-offset, it is checked against some value.

However, the following checks are off-by-one value (`16 ** 3 → (2 ** (3 * 4)) - 1`), which would lead to an overflow by one bit into the next position, if

these maximum values would've been used:

1. `MarketConfigStruct.sol#L38` .
2. `MarketConfigStruct.sol#L54` .
3. `MarketConfigStruct.sol#L72` .
4. `MarketConfigStruct.sol#L90` .
5. `MarketConfigStruct.sol#L108` .
6. `MarketConfigStruct.sol#L129` .
7. `MarketConfigStruct.sol#L149` .

   The following check is impacted as well, however, this field (for maximum trade amount) may take up to 32 bit (`16 ** 8 → (2 ** (8 * 4)) - 1`):
8. `MarketConfigStruct.sol#L171` .

Note: These values may only be set by `MARKET_MGR_ROLE` or `DEFAULT_ADMIN_ROLE` in `MarketValid.setConf()` .

Recommendation: We recommend replacing all mentioned checks with the suggested replacements.

BLX-22
Attacker Can Take Ownership of Implementation Contract

Low Fixed

> **Update**
> The Blex team fixed the issue by implementing the recommendation in commits a48f1a4, c7f82a4 and 5e24371.

File(s) affected: `./contracts/vault/RewardDistributor.sol` , `./contracts/fee/FeeRouter.sol` , `./contracts/market/MarketRouter.sol` , `./contracts/market/Market.sol` , `./contracts/market/OrderMgr.sol` , `./contracts/order/OrderBook.sol` , `./contracts/order/OrderStore.sol` , `./contracts/position/PositionBook.sol` , `./contracts/position/PositionStore.sol` , `./contracts/vault/VaultReward.sol` , `./contracts/vault/CoreVault.sol` , `./contracts/vault/VaultRouter.sol`

Description: Upgradeability often involves two different kinds of contracts: implementation contracts and proxies. This protocol contains implementation contracts that are not automatically initialized. If an implementation contract is not initialized, a malicious user could call `initialize()` on the implementation contract to take ownership of it. If the implementation contract uses delegate calls, it may be possible for an attacker to self-destruct the contract. While no specific

attack vector was found, some contracts, such as `Market` make use of delegate calls, so the risk is heightened.

Recommendation: Add a constructor to the implementation contracts, which calls `_disableInitializers()` to auto-initialize the contract and block a malicious user from taking control of it.

BLX-23

`FREEZER_ROLE` Not Revoked From Previous `VaultRouter`

• Informational Fixed

> **Update**
> Fixed in commit e5aa988 by removing the `FREEZER_ROLE` from the previous `vaultRouter`, as suggested.

File(s) affected: `./contracts/vault/CoreVault.sol`

Description: In the `CoreVault` contract, the address held in the state variable `vaultRouter` is granted the `ROLE_CONTROLLER` and `FREEZER_ROLE` privileged roles. When updating the `vaultRouter` state variable via the function `CoreVault.setVaultRouter()`, only the `ROLE_CONTROLLER` is revoked from the previous `VaultRouter` contract. Thus, the previous `vaultRouter` is still granted the `FREEZER_ROLE` even after updating to a new `VaultRouter` contract.

Recommendation: Revoke the `FREEZER_ROLE` from the previous `vaultRouter` address in the function `CoreVault.setVaultRouter()`.

BLX-24

Unlocked Pragma

• Informational Acknowledged

> **Update**
> The Blex team acknowledged the issue and provided the following explanation:
>
> > Hardhat has been locked at version 0.8.17, and the version configuration is as expected.

File(s) affected: `all`

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.8.*`. The caret (`^`) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, we recommend to remove the caret to lock the file onto a specific Solidity version.

BLX-25 Upgradability

• Informational Acknowledged

> **Update**
> The Blex team acknowledged the issue and indicated that they will move to timelocked upgrades in the future.

File(s) affected: `./contracts/market/OrderMgr.sol`, `./contracts/market/PositionAddMgr.sol`, `./contracts/market/PositionSubMgr.sol`, `./contracts/ac/AcUpgradable.sol`, `./contracts/fee/FeeRouter.sol`, `./contracts/market/MarketRouter.sol`, `./contracts/market/Market.sol`, `./contracts/order/OrderBook.sol`, `./contracts/order/OrderStore.sol`, `./contracts/position/PositionBook.sol`, `./contracts/position/PositionStore.sol`, `./contracts/position/PositionStruct.sol`, `./contracts/referral/Referral.sol`, `./contracts/utils/TransferHelper.sol`, `./contracts/utils/EnumerableValues.sol`, `./contracts/utils/Calc.sol`, `./contracts/vault/VaultReward.sol`, `./contracts/vault/CoreVault.sol`, `./contracts/vault/VaultRouter.sol`, `./contracts/vault/RewardDistributor.sol`

Description: While upgradability is not a vulnerability in itself, users should be aware that many contracts in the codebase can be upgraded at any given time. This audit does not guarantee the behavior of future contracts that may be upgraded to.

Recommendation: The fact that the contract can be upgraded and reasons for future upgrades should be communicated to users beforehand.

BLX-26 Clone-and-Own

Informational | Acknowledged

> **Update**
> The Blex team acknowledged the issue and provided the following explanation:
>
> > The suggestion is good, but considering historical reasons, direct replacement will result in a series of conflicts (so it will only be replaced after new development or refactoring in the future)

File(s) affected: `./contracts/openzeppelin/*`

Description: The clone-and-own approach involves copying and adjusting open source code at one's own discretion. From the development perspective, it is initially beneficial as it reduces the amount of effort. However, from the security perspective, it involves some risks as the code may not follow the best practices, may contain a security vulnerability, or may include intentionally or unintentionally modified upstream libraries.

Recommendation: Rather than the clone-and-own approach, a good industry practice is to use a package manager (e.g., npm) for handling library dependencies. This eliminates the clone-and-own risks yet allows for following best practices, such as, using libraries. If the file is cloned anyway, a comment including the repository, commit hash of the version cloned, and the summary of modifications (if any) should be added. This helps to improve traceability of the file.

1. Consider using `ERC20Upgradable.sol` from Open Zeppelin, as it already utilizes an `initialize()` function rather than a constructor.
2. Consider using `ERC4626Upgradable.sol` from Open Zeppelin, as it already utilizes an `initialize()` function rather than a constructor.

BLX-27 `maxUserNetSizeLimit` Can Be Circumvented

Informational | Acknowledged

> **Update**
> The Blex team acknowledged the issue and indicated that this is the intended design.

File(s) affected: `./contracts/market/GlobalValid.sol`

Description: The `maxUserNetSizeLimit` storage variable restricts how much one side (long vs. short) can be relative to the other for a particular user. Perhaps the intention was to ensure one side is not imbalanced for a particular user. However, a user can use another address to circumvent this limit. Also, the limit may not be desirable, as it restricts the user from rebalancing the protocol. For example, the protocol may have too large a long position, but the user is restricted from creating a large enough short position to balance this due to `maxUserNetSizeLimit`.

Recommendation: Consider removing `maxUserNetSizeLimit` as it can be circumvented and may not be desirable.

BLX-28
Critical Roles Not Granted Upon Deployment

Informational | Acknowledged

> **Update**
> The Blex team acknowledged the issue and provided the following explanation:
>
> > We will authorize relevant roles through scripts during deployment

File(s) affected: `./contracts/fee/FeeVault.sol`

Description: Many of the functions in the codebase have access control to restrict non-authorized callers in the form of privileged roles. In some contracts, the privileged roles needed to access its critical functions are not granted upon deployment. Therefore, the deployer, granted the `DEFAULT_ADMIN_ROLE`, must manually grant privileged roles by calling the `grantRole()` function in the desired contract. This introduces the risk of human error, where a privileged role is granted to the incorrect address. The following privileged roles are not granted upon deployment of their respective contract:

1. `FeeVault`
    1. The `FundFee` contract is not granted the `ROLE_CONTROLLER`, needed to call the function `updateGlobalFundingRate()`.
    2. The `FeeRouter` contract is not granted the `WITHDRAW_ROLE`, needed to call the function `withdraw()`.

Recommendation: Refactor the constructor or `initialize()` function in the aforementioned contracts to grant the critical roles upon deployment.

BLX-29
Risky ERC20 Token Transfer Design

Informational Acknowledged

> **Update**
> The Blex team has acknowledged the issue. They will not make any changes in this version but will consider it in the next one.

File(s) affected: `./contracts/fee/FeeRouter.sol`

Description: In the function `FeeRouter.collectFees()`, fees are collected by transferring ERC20 tokens from the caller to the `FeeVault` contract. While the `collectFees()` function does take in the argument `fees`, which is an array of fee amounts, the actual transfer amount is based on the ERC20 allowance that the caller has given to the `FeeRouter` contract. Therefore, a caller could pass any transfer amount to the `collectFees()` function but reset the allowance for the `FeeRouter` contract to zero. Thus, the `collectFees()` function will not transfer in any tokens but will still emit an event that contains the incorrect fee amount.

Recommendation: The `collectFees()` function should `transferFrom()` the fee amount specified in the `fees` array instead of adjusting for the allowance the caller has given the `FeeRouter` contract.

BLX-30
`delegatecall()` to Untrusted Contract

Informational Acknowledged

> **Update**
> The Blex team acknowledged the issue and provided the following explanation:
>
> > No changes will be made. The primary role of the `Market` is to act as a proxy

File(s) affected: `./contracts/market/Market.sol`

Description: `delegatecall()` is an operation that calls another contract and allows it to execute some arbitrary code and manipulate the storage of the current contract. If the called contract is not trusted, it can lead to dangerous attacks.

The `Market` contract uses `delegatecall()` when calling the `OrderMgr`, `PositionAddMgr`, and `PositionSubMgr` contracts. Each of the three aforementioned contract addresses can be updated by the `MARKET_MGR_ROLE`. If the contract addresses are updated to malicious contracts, the `Market` contract will execute potentially dangerous in the malicious contract via `delegatecall()`.

Recommendation: We recommend against delegating calls to untrusted contracts since they may pose severe security risks. If `delegatecall()` must be used, consider using immutable address state variables for the target contracts.

BLX-31
Gas Inefficient Design

• Informational ⓘ Mitigated

> **Update**
> Mitigated in commit ec83a73 by fixing point 1. Points 2 and 3 cannot be fixed since the contracts have already been deployed, so the state cannot be moved to an enumerable set.

File(s) affected: `./contracts/fee/FeeRouter.sol`, `./contracts/market/Market.sol`

Description:

1. The function `FeeRouter._isMarket()` checks if a given `Market` contract address is registered in the `MarketFactory` contract. This is done by retrieving an array of all registered `Market` contract addresses via the function `MarketFactory.getMarkets()`. After the array of `Market` contract addresses is returned, the array is iterated over, and each element is checked against the given `Market` contract address.

   As the number of `Market` contracts registered in the `MarketFactory` contract increases, the gas costs will also increase. Suppose the number of registered `Market` contracts becomes sufficiently large, and the desired `Market` address is towards the end of the array. In that case, denial-of-service is possible, as the transaction could run out of gas and fail.
2. In the `Market` contract, the function `addPlugin()` iterates over every element in the `plugins` array to validate that the given plugin address is not duplicated. As the size of the `plugins` array increases, the gas costs will also increase. If the `plugins` array becomes sufficiently large, denial-of-service is possible, as the transaction could run out of gas and fail.
3. In the `Market` contract, the function `removePlugin()` iterates over every element in the `plugins` array until the given plugin address is found. As the size of the `plugins` array increases, the gas costs will also increase. If the `plugins` array becomes sufficiently large and the desired plugin address is

towards the end of the array, denial-of-service is possible, as the transaction could run out of gas and fail.

Recommendation:

1. In the `MarketFactory` contract, use an enumerable set rather than an array for the `markets` state variable. The enumerable set can determine if a given `Market` contract address is contained within the set without iterating over each element in the array. (**Update:** Fixed)
2. Create an enumerable set for the plugin contract addresses in the `Market` contract. The enumerable set can determine if a given plugin contract address is contained within the set without iterating over each element in the array.
3. Create an enumerable set for the plugin contract addresses in the `Market` contract. The enumerable set can remove a given plugin contract address without iterating over each element in the array.

BLX-32

`VAULT_MGR_ROLE` Role Holder Can Embezzle Rewards

Informational Acknowledged

> **Update**
> The Blex team acknowledged the issue and provided the following explanation:
>
> > The role can indeed prevent any rewards from being issued during a certain period of time in this way We will not make any changes in this version but will consider it in the next version.

File(s) affected: `./contracts/vault/RewardDistributor.sol`, `./contracts/oracle/FastPriceFeed.sol`

Description: The `VAULT_MGR_ROLE` role holder of contract `RewardDistributor.sol` may embezzle rewards in two ways:

1. Setting the tokens per interval to zero by calling `RewardDistributor.setTokensPerInterval(0)`, leading to no future rewards being distributed.
2. Calling `updateLastDistributionTime()` some time after `distribute()` has been called, leading to all otherwise accumulated rewards between that time frame to being disregarded.

In a similar fashion, `MANAGER_ROLE` of the `FastPriceFeed.sol` contract can influence the price by arbitrarily calling `setLastUpdatedAt()`, where the reason for this function is unclear.

In both cases the functions don't emit an event.

Since in both cases the respective roles have much more direct control over the contract, this issue is of informational severity and seeks to clarify the existence of said update functions.

Recommendation: Clarify and/or document in public-facing documentation the reason for the existence of function `updateLastDistributionTime()`. And clarify and document in public-facing documentation why function `setTokensPerInterval()` accepts `0` tokens per interval as a valid parameter and/or add a check to prevent that parameter. Similarly for `FastPriceFeed.setLastUpdatedAt()`.

BLX-33
Unnecessary Use of `SafeMath` in Solidity 0.8.x

• Informational Fixed

Update
Fixed in commit ec83a73 by removing the import and use of the `SafeMath` library, as suggested.

File(s) affected: `./contracts/vault/VaultReward.sol`, `./contracts/market/PositionSubMgr.sol`

Description: Solidity 0.8.x has a built-in mechanism for dealing with overflows and underflows. There is no need to use the `SafeMath` library (it only increases gas usage).

Recommendation: We recommend against using `SafeMath` in Solidity 0.8.x.

BLX-34
Overflow of Order IDs

• Informational Acknowledged

Update
The Blex team acknowledged the issue and indicated that this is the intended design.

File(s) affected: `./contracts/order/OrderBook.sol`

Description: The function `OrderBook.add()` generates a new order ID by calling `OrderStore.generateID()`. However, before assigning it to the new order it

casts the otherwise `uint256` -typed value down to `uint64` using the primitive/unsafe cast operation `uint64()`:

```
    _order.orderID = uint64(
          (_vars[i].isOpen ? openStore :
closeStore).generateID(
             _order.account
          )
    );
```

Should an account exceed `type(uint64).max` (`18446744073709551615`) orders, the next order would overflow.

A similar overflow is possible in `OrderBook.remove()`.

Recommendation: Consider using instead a `SafeCast` -library to prevent the protocol from ever reaching that number or consider widening the type of `Order.Props.orderID` as well to `uint256` or a type bigger than `uint64`, if deemed necessary.

BLX-35
Invalid

`MarketConfigStruct.VALID_DECREASE_BIT_POSITION` **Overlaps Into** `DECREASE_NUM_LIMIT_BIT_POSITION`

Informational Fixed

> **Update**
> Fixed in commit 902f7ce by increasing the position value to `120 + 4 * 3`, as suggested.

File(s) affected: `./contracts/market/MarketConfigStruct.sol`

Description: Bit position `MarketConfigStruct.VALID_DECREASE_BIT_POSITION` (`120 + 4`) is used to write the enable valid decrease flag from a market config struct using `setEnableValidDecrease()`.

However, this bit position is with `120 + 4` too small and should've been `120 + 4 * 3`, in order to surpass the `DECREASE_NUM_LIMIT_BIT_POSITION` slot.

This would lead to overlapping writes to the decrease order limit field (`getDecrOrderLmt()`).

Note: This flag is never explicitly written, but read in `MarketValid.sol#L250`. While the write function is impacted by the wrong offset, the read function is not.

Recommendation: We recommend adjusting the bit position as described above.

BLX-36
Unchecked return values

Informational Acknowledged

> **Update**
> The Blex team confirmed that they are using OpenZeppelin version 4.8.1 of the `AccessControl` contract. Thus, there are no return values to be checked from the `_grantRole()` and `revokeRole()` functions. The team further confirmed that the return value from `FastPriceFeed.sol#L212:_setLastUpdatedValues(timestamp);` does not need to be checked.

File(s) affected: `./contracts/ac/AcUpgradable.sol`, `./contracts/position/PositionStore.sol`, `./contracts/market/MarketFactory.sol`, `./contracts/vault/CoreVault.sol`, `./contracts/order/OrderStore.sol`, `./contracts/oracle/FastPriceFeed.sol`

Description: To ensure correct behavior it is advised to always check the return value of function calls, if they do return a value. In this regard, consider the following cases:

1. `AcUpgradable.sol#L96`.
2. `PositionStore.sol#L51`.
3. `PositionStore.sol#L53`.
4. `MarketFactory.sol#L145`.
5. `MarketFactory.sol#L147`.
6. `MarketFactory.sol#L148`.
7. `MarketFactory.sol#L149`.
8. `MarketFactory.sol#L154`.
9. `MarketFactory.sol#L163`.
10. `MarketFactory.sol#L168`.
11. `MarketFactory.sol#L173`.
12. `CoreVault.sol#L82`.
13. `CoreVault.sol#L83`.
14. `OrderStore.sol#L38`.
15. `OrderStore.sol#L40`.
16. `FastPriceFeed.sol#L212`.

Recommendation: Consider adding checks for the above listed return values.

BLX-37
Test Code Mixed with Production Code

Informational Fixed

> **Update**
> Fixed in commit ef41230 by removing the test-related lines, as suggested.

File(s) affected: `./contracts/vault/CoreVault.sol`, `./contracts/vault/VaultReward.sol`

Description: For security purposes and gas-optimization it is advised not to mix test code with production code. In this regard, consider the following instances:

1. `CoreVault.sol#L14` .
2. `CoreVault.sol#L269-L270` .
3. `VaultReward.sol#L14` .

Recommendation: We recommend removing the mentioned code and imports.

BLX-38
Application Monitoring Can Be Improved by Emitting More Events

Informational Mitigated

> **Update**
> Mitigated in commit `20f5b4072f93e2668442e1f40ef061b20688c956` by fixing most of the listed issues.

File(s) affected: `./contracts/position/PositionStore.sol`, `./contracts/order/OrderStruct.sol`, `./contracts/order/OrderStore.sol`, `./contracts/oracle/ChainPriceFeed.sol`, `./contracts/vault/RewardDistributor.sol`, `./contracts/oracle/FastPriceFeed.sol`, `./contracts/oracle/Price.sol`, `./contracts/fee/FeeRouter.sol`, `./contracts/vault/CoreVault.sol`, `./contracts/vault/VaultReward.sol`, `./contracts/market/GlobalValid.sol`, `./contracts/market/MarketConfigStruct.sol`,

`./contracts/market/MarketValid.sol`,
`./contracts/market/MarketRouter.sol`,
`./contracts/market/Market.sol`,
`./contracts/market/MarketFactory.sol`

Description: It is a good practice to emit events to validate the proper deployment and initialization of the contracts. Also, any important state transitions can be logged, which is beneficial for monitoring the contract and tracking eventual bugs or hacks. Below, we present a non-exhaustive list of events that could be emitted to improve application management:

1. `PositionStore.set()`: `globalPositions`.
2. `PositionStore.remove()`: `globalPositions`.
3. `OrderStruct.updateTime()`: `_order.updatedAtBlock`.
4. `OrderStore.add()`: `orders[]`, `orderKeys`, `orderNum[]` and `ordersByAccount[]`. (**Update:** Mitigated. Changes to `orderNum[]` and `key` not emitted)
5. `OrderStore.set()`: `orders[]`. (**Update:** Fixed)
6. `OrderStore._remove()`: `orders[]`, `orderKeys`, `orderNum[]` and `ordersByAccount[]`. (**Update:** Mitigated. `orderNum[]` and `key` not emitted)
7. `OrderStore.delByAccount()`: `ordersByAccount[]`. (**Update:** Fixed)
8. `ChainPriceFeed.setSampleSpace()`: `sampleSpace`. (**Update:** Fixed)
9. `ChainPriceFeed.setPriceFeed()`: `priceFeeds[]` and `priceDecimals[]`. (**Update:** Fixed)
10. `RewardDistributor.updateLastDistributionTime()`: `lastDistributionTime`. (**Update:** Fixed)
11. `RewardDistributor.setTokensPerInterval()`: `lastDistributionTime`.
12. `FastPriceFeed.setPriceFeed()`: `chainPriceFeed`. (**Update:** Fixed)
13. `FastPriceFeed.setMaxTimeDeviation()`: `maxTimeDeviation`. (**Update:** Fixed)
14. `FastPriceFeed.setPriceDuration()`: `priceDuration`. (**Update:** Fixed)
15. `FastPriceFeed.setMaxPriceUpdateDelay()`: `maxPriceUpdateDelay`. (**Update:** Fixed)
16. `FastPriceFeed.setSpreadBasisPointsIfInactive()`: `spreadBasisPointsIfInactive`. (**Update:** Fixed)
17. `FastPriceFeed.setSpreadBasisPointsIfChainError()`: `spreadBasisPointsIfChainError`. (**Update:** Fixed)
18. `FastPriceFeed.setMinBlockInterval()`: `minBlockInterval`. (**Update:** Fixed)
19. `FastPriceFeed.setIsSpreadEnabled()`: `isSpreadEnablede`. (**Update:** Fixed)
20. `FastPriceFeed.setLastUpdatedAt()`: `lastUpdatedAt`. (**Update:** Fixed)
21. `FastPriceFeed.setMaxDeviationBasisPoints()`: `maxDeviationBasisPoints`. (**Update:** Fixed)
22. `FastPriceFeed.setMaxCumulativeDeltaDiffs()`: `maxCumulativeDeltaDiffs[]`. (**Update:** Fixed)

23. `FastPriceFeed.setPriceDataInterval()` : `priceDataInterval` . (**Update:** Fixed)
24. `FastPriceFeed.setTokens()` : `tokens` and `tokenPrecisions` . (**Update:** Fixed)
25. `FastPriceFeed._setLastUpdatedValues()` : `lastUpdatedAt` and `lastUpdatedBlock` . (**Update:** Fixed)
26. `FastPriceFeed._setPriceData()` : `priceData[]` .
27. `Price.setAdjustment()` : `isAdjustmentAdditive[]` , `adjustmentBasisPoints[]` and `lastAdjustmentTimings[]` . (**Update:** Fixed)
28. `Price.setFastPriceEnabled()` : `isFastPriceEnabled` . (**Update:** Fixed)
29. `Price.setGmxPriceFeed()` : `gmxPriceFeed` . (**Update:** Fixed)
30. `Price.setFastPriceFeed()` : `fastPriceFeed` . (**Update:** Fixed)
31. `Price.setChainPriceFeed()` : `chainPriceFeed` . (**Update:** Fixed)
32. `Price.setIsGmxPriceEnabled()` : `isGmxPriceEnabled` . (**Update:** Fixed)
33. `Price.setSpreadBasisPoints()` : `spreadBasisPoints[]` . (**Update:** Fixed)
34. `Price.setSpreadThresholdBasisPoints()` : `spreadThresholdBasisPoints` . (**Update:** Deprecated, as function was removed)
35. `Price.setMaxStrictPriceDeviation()` : `maxStrictPriceDeviation` . (**Update:** Fixed)
36. `Price.setStableTokens()` : `strictStableTokens[]` . (**Update:** Fixed)
37. `FeeRouter.setFeeVault()` : `feeVault` . (**Update:** Fixed)
38. `FeeRouter.setFundFee()` : `fundFee` . (**Update:** Fixed)
39. `FeeRouter.setFeeAndRates()` : Wrongfully emitting twice the new rate. Set instead `uint256 _old = feeAndRates[market][i]` in L95. (**Update:** Fixed)
40. `FeeRouter.collectFees()` : Emitting an unused variable `fees` in event `UpdateFee()` . Further, the event name is misleading, as the function rather transfers any pending token allowance from the caller (`onlyFeeController` -modifier) to the `feeVault` contract.
41. `CoreVault.setVaultRouter()` : `vaultRouter` . (**Update:** Fixed)
42. `VaultReward.updateRewardsByAccount()` : `lpEarnedRewards[]` , `averageStakedAmounts[]` , `claimableReward[]` and `previousCumulatedRewardPerToken[]` . (**Update:** Mitigated)
43. `GlobalValid.setMaxSizeLimit()` : `maxSizeLimit` . (**Update:** Fixed)
44. `GlobalValid.setMaxNetSizeLimit()` : `maxNetSizeLimit` . (**Update:** Fixed)
45. `GlobalValid.setMaxUserNetSizeLimit()` : `maxUserNetSizeLimit` . (**Update:** Fixed)
46. `GlobalValid.setMaxMarketSizeLimit()` : `maxMarketSizeLimit[]` . (**Update:** Fixed)
47. `MarketConfigStruct.setMinSlippage()` : `minSp` .
48. `MarketConfigStruct.setMinSlippage()` : `minSp` .
49. `MarketConfigStruct.setMaxSlippage()` : `minSp` .
50. `MarketConfigStruct.setMinLev()` : `minSp` .

51. `MarketConfigStruct.setMaxLev()` : `minSp` .
52. `MarketConfigStruct.setMinPay()` : `minSp` .
53. `MarketConfigStruct.setMinCollateral()` : `minSp` .
54. `MarketConfigStruct.setDecrOrderLmt()` : `minSp` .
55. `MarketConfigStruct.setMaxTradeAmount()` : `minSp` .
56. `MarketConfigStruct.setAllowClose()` : `allow` .
57. `MarketConfigStruct.setEnableValidDecrease()` : `allow` .
58. `MarketConfigStruct.setAllowOpen()` : `allow` .
59. `MarketConfigStruct.setDecimals()` : `_decimals` .
60. `MarketValid.setConfData()` : `conf` . (**Update:** Fixed)
61. `MarketRouter.updatePositionBook()` : `positionBooks` and `pbs[]` . (**Update:** Fixed)
62. `MarketRouter.setIsEnableMarketConvertToOrder()` : `isEnableMarketConvertToOrder` . (**Update:** Fixed)
63. `MarketRouter.addMarket()` : `markets` , `positionBooks` and `pbs[]` . (**Update:** Mitigated. State variable change to `_positionBook` not emitted)
64. `MarketRouter.removeMarket()` : `markets` , `positionBooks` and `pbs[]` . (**Update:** Mitigated. State variable change to `_positionBook` not emitted)
65. `Market.addPlugin()` : `plugins[]` . (**Update:** Fixed)
66. `Market.removePlugin()` : `plugins[]` . (**Update:** Fixed)
67. `Market._setOrderBook()` : `orderBookLong` and `orderBookShort` . (**Update:** Fixed)
68. `Market.setPositionBook()` : `positionBook` . (**Update:** Fixed)
69. `Market.setMarketValid()` : `marketValid` . (**Update:** Fixed)
70. `Market.setPositionMgr()` : `positionAddMgr` and `positionSubMgr` . (**Update:** Fixed)
71. `Market.setOrderMgr()` : `orderMgr` . (**Update:** Fixed)
72. `MarketFactory.remove()` : `markets` . (**Update:** Fixed)
73. `MarketFactory.create()` : `markets` . (**Update:** Fixed)

Recommendation: Consider emitting the events.

BLX-39
Over-Engineered and Highly Coupled Contracts

- **Undetermined**
Acknowledged

> **Update**
> The Blex team acknowledged the issue and provided the following explanation:
>
> > The three contract designs are considered, and various strategies and solutions for future application in traditional

> markets are reserved in the contract without modification.

File(s) affected: `./contracts/fee/FeeRouter.sol`, `./contracts/fee/FeeVault.sol`, `./contracts/fee/FundFee.sol`, `./contracts/market/Market.sol`, `./contracts/market/MarketValid.sol`

Description: Excessive complexity and highly coupled code can have several negative consequences that can impact the security, maintainability, and readability of smart contracts. Over-engineered code refers to writing software that is overly complex beyond what is necessary to meet the requirements and objectives of the project. Highly coupled code refers to code where different components are tightly interconnected and dependent on each other. Here are some consequences of such an approach:

- **Security Vulnerabilities**: Tight coupling can create hidden dependencies and unexpected interactions between components, making it easier for security vulnerabilities to arise. Changes to one part of the code might inadvertently expose vulnerabilities in other parts.
- **Complexity**: The more complex the code, the more likely it is to contain bugs. Unintended interactions between over-engineered components can lead to difficult-to-diagnose issues requiring significant effort.
- **Difficulty in Maintenance**: Highly coupled code makes it challenging to make changes or updates to individual components without affecting other parts of the codebase. This can lead to unintended side effects and make the codebase more error-prone.
- **Gas Consumption**: Contracts that are tightly coupled will consume greater amounts of gas due to the higher number of external calls.
- **Code Readability and Understanding**: Code that is tightly interconnected can be difficult to understand, especially when technical documentation and code comments are lacking. This can lead to slower comprehension and an increased risk of overlooked bugs.

Specifically, there are instances in the codebase where related global state variables are split across two or more contracts, leading to disconnected functionality and complex workflows.

1. The fee contracts (`FeeRouter`, `FeeVault`, and `FundFee`) are highly coupled and bloated with external calls to each other. This is because the fee global state is unnecessarily split across all three contracts. `FeeRouter` holds the state for each `Market` contract's fee rates, `FeeVault` holds the state for each `Market` contract's funding rates, and `FundFee` holds the state for each `Market` contract's funding interval. For example, the call trace when executing `FeeRouter.updateCumulativeFundingRate()` is:
    1. `FeeRouter.updateCumulativeFundingRate()`
    2. `FundFee.updateCumulativeFundingRate()`
    3. `FeeVault.updateGlobalFundingRate()`

2. The `Market` and `MarketValid` contracts have a one-to-one relationship, where the `MarketValid` contract holds the relevant global state for a given `Market` contract. Therefore, the `Market` contract must make external calls to the `MarketValid` contract to validate position and order updates, which results in excessive gas costs and reduced readability.

Recommendation:

1. Consider combining the three fee contracts `FeeRouter`, `FeeVault`, and `FundFee` into one contract. This will reduce deployment complexity and potential for misconfiguration.
2. Consider combining the `Market` and `MarketValid` contracts into one contract. This will reduce deployment complexity and potential for misconfiguration.
3. We recommend changing the design to adhere more to the diamond pattern (ERC-2535), having one storage contract and multiple facet contracts, that purely hold logic and no state.
4. Further, we recommend redesigning the privileged roles to be defined and managed within a dedicated contract, to simplify role management.

BLX-40
Inconsistent Use of the `minRateLimit`

• **Undetermined** Acknowledged

> **Update**
> The Blex team acknowledged the issue and indicated that the functionality is as designed.

File(s) affected: `./contracts/fee/FundFee.sol`

Description: In `FundFee.updateCumulativeFundingRate()`, the first time a funding rate is assigned for a Market, the global funding rate is assigned all 0s instead of the `minRateLimit`. This does not enforce the `minRateLimit` for a market that is processing the first orders.

Recommendation: Consider whether the `minRateLimit` is supposed to be enforced during this time and update the global funding rate to the `minRateLimit` instead of 0.

BLX-41
Missing Reentrancy Guards

• **Undetermined** Acknowledged

Update

The Blex team acknowledged the issue and decided not to add reentrancy guards since no specific exploits were identified. We recommend adding reentrancy guards to reduce attack surface even if a definite exploit has not been found.

File(s) affected: `./contracts/vault/VaultReward.sol`

Description: The comments of `VaultReward.updateRewardsByAccount()` suggest that the function should be called without reentry; however, there is no reentrancy modifier present.

Recommendation: We recommend including the ReentrancyGuard. Further, we recommend adding reentrancy guards to all externally callable functions that interact with external contracts.

BLX-42
Keepers Are A Single Point Of Failure

• Undetermined Acknowledged

Update

The Blex team indicated that the liquidation function on the Market contract can be called by another contract, which will be accessible by any user.

File(s) affected: `./contracts/market/Market.sol`

Description: Keepers play a critical role in the Blex protocol. Only approved keepers can liquidate positions or execute valid orders. If, for some reason, the approved keepers do not perform these actions promptly, there can be many negative consequences. For example, if liquidation stopped, this would result in LPs losing any potential profits. This could happen if the keepers were attacked in some way or if they were maliciously used.

Recommendation: Design the protocol so that anyone can execute liquidation and valid orders.

BLX-43
Orders Do Not Support Expiry Time

• Undetermined Acknowledged

> **Update**
> The Blex team acknowledged the issue. They will not implement expiry times as users can cancel their orders manually at any time.

Description: Within Blex, orders are executed asynchronously. The trader may place the order, but the keeper will execute the order in a separate transaction. The keeper will likely execute the order quickly after the conditions for the order have been fulfilled. However, the keeper may execute the order much later than it should. Because market conditions can change drastically over time, it would make sense to support an order expiry time so that it cannot be executed after that time.

Recommendation: Consider implementing an expiry time for orders.

BLX-44

`FastPriceFeed` Relies on Centralized Updater

• **Undetermined** Acknowledged

> **Update**
> The Blex team acknowledged the issue. The reason for having the `FastPriceFeed` is to have a price feed that updates faster than Chainlink oracles. Currently, the only way to update the `FastPriceFeed` is for a privileged account to call the appropriate update function. The team will consider if less centralized designs are possible in the future.

File(s) affected: `./contracts/oracle/FastPriceFeed.sol`

Description: Only a privileged account with the `PRICE_UPDATE_ROLE` role may update the prices within the `FastPriceFeed` contract. If the system or bot leveraging this account becomes unavailable, the price feed will not be updated. Depending on the configuration of the `Price` contract, the price will be either unavailable or immediately shifted to the Chainlink price feed, the GMX price feed, or some variation of the two. Attackers may exploit the differences in prices. It would be ideal if the price feed used would be decentralized and not rely on a privileged updater to set the price.

Recommendation: Consider designing a decentralized version of the fast price feed for future iterations. Carefully monitor the updates of the `FastPriceFeed` to determine if it is not being updated. If so, consider temporarily disabling it until the issue has been resolved.

BLX-45

`PositionStore.set()` and `PositionStore.remove()` Callable by All `ROLE_CONTROLLER` Roles

- **Undetermined**
Acknowledged

> **Update**
> The Blex team acknowledged the issue and indicated that this is the intended design.

File(s) affected: `./contracts/position/PositionStore.sol`

Description: Function `PositionStore.setPositionBook()`, which is only callable by the `DEFAULT_ADMIN_ROLE` role (`onlyAdmin`-modifier), updates the `positionBook` address variable, transfers the `ROLE_CONTROLLER` role to it and emits the event `UpdatePositionBook()`.

The contract further contains functions `set()` and `remove()` for setting and removing positions, which can only be called by the `ROLE_CONTROLLER` role.

However, as functions `set()` and `remove()` are only guarded by the `onlyAdmin`-modifier they can be called by all addresses that hold the `ROLE_CONTROLLER` role, which can be granted by `DEFAULT_ADMIN_ROLE` via `grantRole()`, without the need of calling `setPositionBook()`.

While no vulnerability in and by itself, it needs to be clarified if this is intended behaviour, as multiple addresses may access these functions, without triggering the `UpdatePositionBook()` event.

Recommendation: Clarify if this is intended behaviour, or if necessary, further constrain functions `set()` and `remove()` to be only callable by `positionBook`.

BLX-46

Incorrect Accounting of Order Numbers

- **Undetermined**
Acknowledged

> **Update**
> The Blex team acknowledged the issue and indicated that this is the intended design.

File(s) affected: `./contracts/order/OrderStore.sol`

Description: Function `OrderStore.add()` always increments the order counter (`orderNum`) for the given account. However, it does not check whether the order is actually new or just updating an existing one, potentially leading to an incorrect number of orders per account.

Recommendation: We recommend adding a check to only add new orders (i.e. `orders[order.getKey()]` must be empty) or only increment the `orderNum` counter, if the order is new and not an update of an existing order.

BLX-47

`FastPriceFeed.setPricesWithBits()` Always Updates All Up to 8 Tokens

- **Undetermined** Acknowledged

> **Update**
> The Blex team acknowledged the issue and indicated that this is the intended design.

File(s) affected: `./contracts/oracle/FastPriceFeed.sol`

Description: Function `FastPriceFeed.setPricesWithBits()` , in particular `_setPricesWithBits()` , always will update all tokens (up to eight) with the price information provided in parameter `_priceBits` . Providing a zero bitmask of 32 zeroes would overwrite that tokens price with the smallest unit of `PRICE_PRECISION` (which would further be processed within `_setPrice()`).

As the function is not used inside the scope of this audit, it is unclear as to how it will be used.

Recommendation: Clarify if this is intended behaviour and/or consider adding another parameter that specifies the number of to-be-updated tokens.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if

exploited, or are reasonably likely to lead to moderate financial impact.

- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Code Documentation

1. The following typographical errors have been noted:
    1. `AcUpgradable.sol` → `AcUpgradeable.sol`.
    2. `TransferHelper.sol#L17`: `VAULR_REWARD_BASE_PRECISION` → `VAULT_REWARD_BASE_PRECISION`.
    3. `PositionAddMgr.sol#L235`: `_transationsFees()` → `_transactionFees()`. (**Update:** Fixed)
    4. `MarketDataTypes.sol#L182`: `totoalFees()` → `totalFees()`.
    5. `MarketValid.sol#L406`: `_totoalFees` → `_totalFees`.
    6. `FastPriceFeed.sol#L33`: `isSpreadEnablede` → `isSpreadEnabled`.
    7. `FastPriceFeed.sol#L94`: `minter` → `admin`.
    8. `Price.sol#L48 and L94`: `PriceFeed` → `Price`.
    9. `FundFee.sol#L18`: `BASIS_INTERVAL_HOU` → `BASIS_INTERVAL_HOUR`.
    10. `FeeRouter.sol#L61`: `feeRouter` → `FeeRouter`.
    11. `CoreVault.sol#L226`: `vault` → `CoreVault`.
    12. `VaultRouter.sol#L55`: `MarketSetted` → `MarketSet`.
    13. `MarketRouter.sol#L335 and L338`: `USDei` → `USD`.
2. Several `external` / `public` functions are lacking NatSpec documentation. Consider add these to improve code quality and ease further development or external integrations.
3. Contract `ChainPriceFeed.sol` uses `PriceFeed` instead of `ChainPriceFeed` in its revert messages. Clarify if this is intentional or adjust accordingly.

4. Contract `RewardDistributor.sol` states at the following occurences that only the admin can execute the corresponding functions. However, they are only executable by the `VAULT_MGR_ROLE` role holder, not the admin (which is `MANAGER_ROLE` as per `AcUpgradable.sol`):
    1. `RewardDistributor.sol#L52` : `updateLastDistributionTime()` .
    2. `RewardDistributor.sol#L60` : `setTokensPerInterval()` .
5. `FeeRouter.sol#L110` : Wrongfully states that only `the controller can call this function` . Instead, only a market that has the `allowOpen` or `allowClose` flag set, as defined by `getMarkets()` of the `factory` contract, can call this function (`onlyMarket` -modifier).
6. `FeeRouter.sol#L129` : Similarly, states that only `the controller can call this function` . However, a controller (address that holds the `ROLE_CONTROLLER` role) **and** a market, as described above, can both call this function (`onlyFeeController` -modifier).
7. `VaultRouter.sol#L87` : Wrongfully states that the function can only be called by `MARKET_ROLE` . Such a role does not exist and that function (`transferToVault()`) is only callable by `ROLE_CONTROLLER` (`onlyController` -modifier).
8. `VaultReward.sol#L212` : The NatSpec documentation for function `getLPPrice()` states it internally calls `getLPPrice()` of the `coreVault` contract. However, this is not the case, it uses `totalAssets()` and `totalSupply()` .
9. `VaultReward.sol#L244–L246` : States that it returns the USD balance of the **calling** contract. However, it returns the USD balance of the saved `vaultRouter` contract.
10. `MarketValid.isLiquidate()` : Function may return `2` , but this state is not described.
11. Consider having a code comment for each mapping which documents what each key and value represents. For example: `(market => feeType => fee)` .
12. Consider renaming `MarketRouter.updateOrder` to `MarketRouter.addUpdateOrder` or `MarketRouter.upsertOrder()` .
13. Interface names should ideally start with `I` . For example, `MarketCallBackIntl` could be renamed to `IMarketCallBackIntl` .
14. Code comments for the function `VaultReward.getUSDBalance()` state "This function retrieves the USD balance of the contract calling the function, by calling the getUSDBalance function of the vaultRouter contract". However, the `getUSDBalance()` function does **NOT** return the USD balance of the calling contract but instead returns the USD balance of **ALL** vaults registered in the `VaultRouter` .

Adherence to Best Practices

1. For maintainability and to increase gas efficiency it is advised to removed 'dead' or unused code. In this regard, consider the following cases:

1. `Ac.sol` : Unused imports to `AccessControl.sol` , `Ownable.sol` and `Initializable.sol` .
2. `MarketValid.validCollateralDelta()` has unnecessary variable `newCollateral` and unnecessary logic verifying it.
3. `AcUpgradable.sol` : Inherits from `Ownable` but otherwise does not make meaningful use of its features (`onlyOwner` -modifier). The only occurence of the use of that modifier is in contract `MarketReader.sol` , which inherits from `AcUpgradable.sol` . Considering most contracts inherit from `AcUpgradable` (or `Ac.sol`), significant gas could be saved by removing this inheritance and refactor the use in `MarketReader.sol` .
4. `Calc.sum()` : Unnecessary use of `abs()` as the surrounding if-else-clause takes care of checking the signedness of parameter `b` . (**Update:** Fixed)
5. `TransferHelper.IERC20Decimals` : This interface declaration is unnecessary as interface `IERC20` already defines the same `decimals()` function. (**Update:** Fixed)
6. `TransferHelper.transferIn()` : Function is never used.
7. `PositionBook.sol#L26` : Grants `msg.sender` the `MANAGER_ROLE` role. However, this role is unused throughout the contract and can therefore be removed. (**Update:** Fixed)
8. `OrderLib.sol#L5` : Unused import to `OrderStruct.sol` . (**Update:** Fixed)
9. `MarketDataTypes.sol#L177–L178` : Empty else-clause that can be removed. (**Update:** Fixed)
10. `PositionBook.sol#L185` : Unnecessary update of the position `isLong` field.
11. `OrderBook.getExecutableOrdersByPrice()` : Unnecessary double iteration through orders. Consider using only one loop and a dynamically sized array for gas optimization.
12. `RewardDistributor.sol#L9` : Unused import to `Ac.sol` . (**Update:** Fixed)
13. `FastPriceFeed.sol#L95` : Spurious call to `_grantRole(DEFAULT_ADMIN_ROLE, msg.sender)` . The `DEFAULT_ADMIN_ROLE` will be already initialized to `msg.sender` through call to `AcUpgradable._initialize()` during `Ac.sol` constructor execution. (**Update:** Fixed)
14. `FastPriceFeed.sol#L564` and `L568` : Unnecessary declaration and use of local variable `_maxTimeDeviation` . Variable `maxTimeDeviation` can be used directly.
15. `FastPriceFeed.sol#L404` : Unnecessary declaration and use of local variable `index` . Variable `j` can be used directly. (**Update:** Deprecated, as function `_setPriceWithBits()` was removed and replaced with `_setPriceWithBits_2()`)
16. For gas-optimization the check in `FastPriceFeed.sol#L405` may be moved outside the loop and, if necessary, the actual token count can be used as the for-loop variable. (**Update:** Deprecated, as function

`_setPriceWithBits()` was removed and replaced with `_setPriceWithBits_2()`)

17. Consider removing unsued function `FastPriceFeed._setPricesWithBits_2()`, or replace `_setPricesWithBits()` with it. (**Update:** Fixed)
18. Unused storage variable `Price.spreadThresholdBasisPoints` and function `Price.setSpreadThresholdBasisPoints()`. (**Update:** Mitigated, by removing unused function `setSpreadThresholdBasisPoints()`. However, storage variable `spreadThresholdBasisPoints` remains)
19. `FundFee.sol`: Spurious import and inheritance of `Ownable.sol`, since the inherited contract `Ac` (`AcUpgradable`) already inherits from `Ownable`. (**Update:** Mitigated, by removing the inheritance to `Ownable`. However, the unused import to `Ownable.sol` remains)
20. `FeeRouter.getOrderFees()`: Unnecessarily returns a `int256` value and is using `SafeCast`-operation `toInt256()`. However, given order fees can only be positive, the primitive cast operation `int256()` may be used instead to save on gas. Further, consider changing the return type to `uint256`.
21. `CoreVault.sol#L74`: Unnecessary expression `1 *`. (**Update:** Fixed)
22. `VaultRouter.sol#L128`: Unnecessary re-check if `msg.sender` is a market. Such a check is already performed in the used modifier `onlyMarket`. (**Update:** Fixed)
23. `VaultReward.sol#L219`: Unnecessary expression of `1 *`.
24. `VaultReward.apr`: State variable `apr` as well as functions `setAPR()` and `getAPR()` remain unused throughout the codebase. Clarify their use or consider removing them.
25. `GlobalValid.sol#L4`: Unused import of `Ownable`. (**Update:** Fixed)
26. `Market._validSender()`: Unused function. (**Update:** Fixed)

2. Consider moving function `OrderLib.getKey()` into contract `OrderStruct.sol` and re-using it in `OrderStruct.getKey()` for improved maintainability. (**Update:** Mitigated, by making use of `OrderLib.getKey()` in `OrderStruct.getKey()`)
3. Similarly, however with the aim of increasing readability and maintainability, commented-out code should be removed. In this regard, consider the following cases:
    1. `RewardDistributor.sol#L22`. (**Update:** Fixed)
    2. `RewardDistributor.sol#L67-L72`. (**Update:** Fixed)
    3. `OrderMgr.sol#L302`. (**Update:** Fixed)
    4. `FeeRouter.sol#L145`. (**Update:** Fixed)
    5. `VaultReward.sol#L123`. (**Update:** Fixed)
    6. `MarketValid.sol#L340`. (**Update:** Fixed)
    7. `MarketRouter.sol#L243`. (**Update:** Fixed)
    8. `Market.sol#L261-L264`. (**Update:** Fixed)
4. For clarity and readability consider using descriptive variable and function names. In this regard, consider the following instances:

1. `Calc.sol` : The contract contains 2 functions named `sum()` (with different parameters). However, they produce different results. The `sum()` function with parameters `(uint256 a, int256 b)` returns the sum of the two parameters, but taking the absolute value of parameter `b` . The `sum()` function with parameters `(int256 a, uint256 b)` does return the actual sum of both parameters, but casts parameter `b` to a signed integer. Consider renaming the functions to prevent confusing the two (i.e. `sum_abs()` and `sum_cast()`).
2. `TransferHelper.sol` : Function `parseVaultAssetSigned()` albeit being similarly named to `parseVaultAsset()` rather behaves identical to function `formatCollateral()` with the only difference of accepting (and returning) a **signed** `amount` value. Consider renaming `parseVaultAssetSigned()` to `formatCollateralSigned()` to prevent confusion.
3. Contracts `OrderStruct.sol` and `PositionStruct.sol` both define a structure called `Props` . Consider giving them individual names for clearer distrinction.
4. `OrderStruct.setPairKey()` : Rather than setting a pair key (as retrieved from `OrderLib.getKey()`), the function sets the order ID.
5. `FeeRouter.onlyFeeController` : This modifier name indicates that only the controller (`ROLE_CONTROLLER` role) can access such protected functions. However, also markets that have the `allowOpen` or `allowClose` flag set, as defined by `getMarkets()` of the `factory` contract, can call this function. Consider renaming it to i.e. `onlyRoleControllerOrMarket` .
6. `FeeRouter.collectFees()` : Emits an event called `UpdateFee()` . However, instead of updating fees this function rather transfers any pending token allowance from the caller (`onlyFeeController` -modifier, see above) to the `feeVault` contract. Consider renaming it to i.e. `FeesCollected()` .

5. Consider adhering to the official Solidity style guide for consistency. In this regard, consider the following cases:
   1. `TransferHelper.usdDecimals` : Constants should be named with all capital letters with underscores separating words. (**Update:** Fixed)
   2. `MarketStorage.pluginGasLimit` : Constants should be named with all capital letters with underscores separating words. (**Update:** Deprecated, as storage variable was removed)
   3. `CoreVault._transFeeTofeeVault()` : Function names should use mixedCase. (**Update:** Fixed)
6. To facilitate logging it is recommended to index address parameters within events. Therefore the `indexed` keyword should be added to the (other) address parameters in
   1. `FastPriceFeed.UpdatePrice()` .
   2. `FastPriceFeed.PriceData()` .
   3. `FastPriceFeed.MaxCumulativeDeltaDiffExceeded()` .
   4. `MarketRouter.UpdatePosition()` .
   5. `MarketRouter.UpdateOrder()` .

6. `MarketRouter.DeleteOrder()`.
7. `MarketFactory.Create()`.
8. `Referral.SetTraderReferralCode()`.
9. `Referral.SetReferrerTier()`.
10. `Referral.SetReferrerDiscountShare()`.
11. `Referral.RegisterCode()`.
12. `Referral.SetCodeOwner()`.
13. `Referral.GovSetCodeOwner()`.
14. `Referral.IncreasePositionReferral()`.
15. `Referral.DecreasePositionReferral()`.
16. `VaultRouter.MarketSetted()`.
17. `VaultRouter.MarketRemoved()`.
18. `VaultReward.Harvest()`.

7. For increased readability and maintainability it is advised to use enums over boolean values when modeling a status rather than a boolean state. In this regard, consider the following cases:
    1. Several contracts throughout use `true` and `false` when addressing a long or short position. Consider replacing them with enum values (i.e. `enum PositionType { SHORT_POSITION, LONG_POSITION }`):
        1. `PositionBook.sol`.
        2. `OrderBook.sol`.
        3. `OrderStore.sol`.
    2. Similarly, the following contracts have a variable `bool isOpen`, which could be replaced with an enum (i.e. `enum OrderType { OPEN_ORDER, CLOSE_ORDER }`):
        1. `MarketDataTypes.sol`.
        2. `OrderBook.sol`.
        3. `MarketStorage.sol`.
        4. `PositionBook.sol`.
    3. Likewise, boolean values `true` and `false` are used to describe the long and short funding rates and could be replaced with an enum in contract `FeeVault.sol`.
    4. `OrderStruct.sol`: Uses `1` for `triggerAbove` being `true` and `2` for `triggerAbove` being `false`. Consider using an enum or directly a boolean variable instead.
    5. `VaultRouter.sol`: Freeze types `0`, `1` and `2` should be replaced with an enum.
    6. `MarketValid.sol`: Consider using an enum for `busType` in `validCollateralDelta()`.
    7. `MarketValid.sol`: Consider using en enum for the return types of `validateLiquidation()`.
    8. `MarketRouter.sol`: Consider using an enum for category types in `updatePositionCallback()`.
    9. `MarketReader.sol`: Consider re-suing the fee type enum (`getMarket()`).
8. To improve readability and lower the risk of introducing errors when making code changes, it is advised to not use magic constants throughout code, but

instead declare them once (as constant and commented) and use these constant variables instead. Following instances should therefore be changed accordingly:

1. `PositionBook.sol#L474` : `100` . (**Update:** Fixed)
2. `CoreVault.sol#L74` : `100` . (**Update:** Fixed)
3. `OrderBook.sol#L42` : `5` .
4. `FeeRouter.sol#L304` : `100000` . (**Update:** Fixed)
5. `CoreVault.sol#L200` : `5` and `6` (Consider re-using `IFeeRouter.FeeType`). (**Update:** Fixed)
6. `VaultRouter.sol#L221` : `8` . (**Update:** Fixed)
7. `MarketConfigStruct.sol` : Several undocumented constants. (**Update:** Fixed)

9. Remaining `TODO` s should be resolved before deployment. In this regard, consider the following cases:
   1. `OrderStruct.sol#L31` : `96 todo uint96 extra4;` .
   2. `CoreVault.sol#L80` : `fix: CVB-05 | Old vault router is not removed from` ROLE_CONTROLLER `when setting new vault router` . (**Update:** Fixed)
   3. `MarketValid.sol#L177` : `MVB-02 : Incorrect input for "validSize(' function` . (**Update:** Fixed)
   4. `MarketValid.sol#L241` : `MVB-03 : Incorrect Error Message` . Also consider correcting the corresponding error message. (**Update:** Fixed)
   5. `OrderMgr.sol#L283` : `OMB-03 COLLATERAL TO BE REFUNDED SHOULD SUBTRACT THE execFee` . (**Update:** Fixed)
10. Function `OrderStore.filterKeys()` does not validate the uniqueness of the given key array `_ordersKeys` , potentially returning a higher count should duplicates exist in the array. While no exploit scenario in the given code base was found, it is advised to keep this limitation in mind in case of future re-factorings, i.e. by adding a corresponding comment.
11. It is according to best practices to have an implementation contract always inherit from its interface. In this regard, consider the following cases:
    1. `FeeVault.sol` .
    2. `FundFee.sol` .
    3. `GlobalValid.sol` . (**Update:** Fixed)
    4. `Market.sol` .
    5. `MarketRouter.sol` .
    6. `ChainPriceFeed.sol` .
    7. `FastPriceFeed.sol` .
    8. `Price.sol` .
    9. `OrderStore.sol` .
    10. `PositionBook.sol` .
    11. `PositionStore.sol` .
    12. `RewardDistributor.sol` .
    13. `VaultReward.sol` .
    14. `VaultRouter.sol` .
12. `FundFee.FEE_RATE_PRECISION` : Re-definition of the variable. Consider re-using `TransferHelper.FEE_RATE_PRECISION` instead. (**Update:** Fixed)

13. Consider using more descriptive error messages (non-exhaustive list):
    1. `GlobalValid.sol#L25` .
    2. `GlobalValid.sol#L35` .
    3. `GlobalValid.sol#L45` .
14. `GlobalValid.sol` : Functions `_getMaxUseableGlobalSize()` , `_getMaxUseableNetSize()` , `_getMaxUseableUserNetSize()` and `_getMaxUseableMarketSize()` could be refactored to save on gas-cost by unifying common code to one function. (**Update:** Fixed)
15. `FeeRouter.collectFees()` does not use the parameter `address account` . It can be removed and all calls should be modified.
16. Private function naming:
    1. `PositionAddMgr.commitIncreasePosition()` . (**Update:** Fixed)
    2. `PositionSubMgr.decreasePositionFromOrder()` . (**Update:** Fixed)
    3. `Market.getPrice()` . (**Update:** Fixed)
    4. `MarketRouter.calculateEquivalentCollateralAmount()` . (**Update:** Fixed)
    5. `OrderBook.setupTriggerAbove()` . (**Update:** Fixed)
    6. `OrderStore.filterOrders()` . (**Update:** Fixed)
    7. `VaultReward.stakedAmounts()` . (**Update:** Fixed)
    8. `VaultRouter.updateFundsUsed()` . (**Update:** Fixed)
17. In `MarketRouter.cancelOrderList()` the check for duplicate orders is an unnecessary costly loop as in `OrderMgr.cancelOrderList()` the processing of the order cancel will not occur if the order is not found. Therefore, if the list is passed with duplicate orders, the first will be processed and the second will be ignored.
18. `_vars.oraclePrice` does not need to be written to in `MarketValid.isLiquidate()` as it is unused in `feeRouter.getFees()` or anywhere else in the execution of this function. (**Update:** Fixed)
19. In `PositionSubMgrLib.calculateNewCollateral()` the second to last condition is excessive as the same value would be returned as long as the `_params.collateralDelta.toInt256() + dPNL - fees > 0` is not less than zero. (**Update:** Fixed)
20. In `FundFee.updateCumulativeFundingRate()` , the lines of code `_lastTime = (_getTimeStamp() / _fundingInterval) * _fundingInterval;` contains unnecessary multiplication and subtracting, and can be simplified to be `_lastTime = _getTimeStamp()` . Further, the function `_getTimeStamp()` simply returns `block. timestamp` gas can be saved by just doing `_lastTime` .
21. `Market.SELECTOR_EXE_ORDER_KEY` should be declared at the top of the file. (**Update:** Fixed)
22. `MarketLib.feeWithdraw()` is an internal function that is not called anywhere in the codebase. Consider if it should remain in the codebase.
23. `PositionAddMgr._execIncreaseOrderKey()` contains a redundant check that `order.account != address(0)` as that is already checked in `Order.validOrderAccountAndID()` . (**Update:** Fixed)
24. `MarketStorage.orderStore()` should be declared after all of the variables in the file. (**Update:** Fixed)

25. `MarketValid.validIncreaseOrder()` does not need to call `validSize()` as `validSize()` contains no logic for if `_isIncrease` is true.
26. `PositionAddMgr._transationFees` contains a spelling error and should be renamed to `_transactionFees()`. (**Update:** Fixed)
27. `PositionBook.constructor()` contains a typo as `factoty` should be spelled `factory`. (**Update:** Fixed)
28. The state variable `CoreVault.vaultReward` should be declared at the top of the file with the rest of the state variables.
29. The parameter `vault` in `VaultRouter.setMarket` is unneeded, as it will immediately be overwritten. (**Update:** Fixed)
30. Since the inherited `Ownable` contract does not have constructor arguments, the `Ac.sol` contract can remove `Ownable()` from the constructor.
31. Remove unused state variables and functions from contracts to increase readability and decrease deployment costs.
    1. `AcUpgradable.sol`
        1. `FEE_DISTRIBUTOR_ROLE` (**Update:** Fixed)
        2. `FEE_MGR_ROLE` (**Update:** Fixed)
    2. `FundFee.sol`
        1. `blankSlot`
    3. `Market.sol`
        1. `_validSender()` (**Update:** Fixed)
    4. `MarketReader.sol`
        1. `getMin()` (**Update:** Fixed)
    5. `FastPriceFeed.sol`
        1. `_setPricesWithBits_2()` (**Update:** Fixed)
32. The modifier `onlyInitOr()` in the `AcUpgradable` contract calculates the daily seconds with `3600 * 24`. To improve readability, replace with `1 day`. (**Update:** Fixed)
33. Rename the modifier `onlyAdmin()` in the `AcUpgradable` contract to `onlyManager()` to avoid confusion between the `DEFAULT_ADMIN_ROLE`. (**Update:** Fixed)
34. Refactor the modifier `onlyUpdater()` in the `AcUpgradable` validates the caller has the `PRICE_UPDATE_ROLE` via the function `hasRole()`. All other modifiers in the `AcUpgradable` contract validate the caller's role via the function `_checkRole()`. Refactor `onlyUpdater()` to remain consistent with the other modifiers. (**Update:** Fixed)
35. All `require`, `assert`, and `revert` statements should include an error message to more easily determine why execution fails and pinpoint potential bugs. An even better approach is to use custom errors rather than error messages, as custom errors are more gas-efficient. The following functions and modifiers contain `require` statements with missing error messages:
    1. `AcUpgradable.onlyUpdater()`. (**Update:** Deprecated)
    2. `AcUpgradable.grantControllerRoleByMarketManager()`. (**Update:** Mitigated)
    3. `Market.receive()`. (**Update:** Mitigated)
    4. `Market.setOrderBooks()`. (**Update:** Mitigated)

5. `Market.setPositionBook()`. (**Update:** Mitigated)
6. `Market.setMarketValid()`. (**Update:** Mitigated)
7. `Market.setPositionMgr()`. (**Update:** Mitigated)
8. `Market.setOrderMgr()`. (**Update:** Mitigated)
9. `MarketRouter.updatePositionBook()`. (**Update:** Mitigated)
10. `RewardDistributor.initialize()`. (**Update:** Mitigated)

36. The contract `FundFee` unnecessarily inherits the `Ownable` contract since the parent contract `Ac` already inherits `Ownable`. Remove the `Ownable` inheritance from the `FundFee` contract. (**Update:** Fixed)
37. Getter functions that only return global state (`block.timestamp`, `msg.sender`, etc.) are unnecessary, gas inefficient, and reduce readability. For example, the `FundFee` contract has the function `_getTimeStamp()`, which only returns `block.timestamp`. Consider removing these functions.
38. The library `GlobalDataTypes` only includes the struct `ValidParams`. Structs can be declared directly in a file without being part of a contract, library, or interface. (**Update:** Fixed)

```
// GlobalDataTypes.sol


struct ValidParams {
    address market;
    uint256 sizeDelta;
    bool isLong;
    uint256 globalLongSizes;
    uint256 globalShortSizes;
    uint256 userLongSizes;
    uint256 userShortSizes;
    uint256 marketLongSizes;
    uint256 marketShortSizes;
    uint256 aum;
}
```

39. In the `Market` contract, the state variable `SELECTOR_EXE_ORDER_KEY` is assigned the function selector for `execOrderKey()`, obtained by hashing the function signature. A better and cleaner approach is to obtain the function selector via `execOrderKey.selector`. (**Update:** Fixed)
40. Each time a contract state variable is read, the data is retrieved via the opcode `SLOAD`, which consumes more gas than retrieving the data from memory (`MLOAD`). Therefore, if it is known that the state of a contract state variable will not change within the execution of a function, and the state variable is read multiple times, it is more gas efficient to cache the state variable and reference the cached data, which will use the `MLOAD` opcode instead. Consider caching state variables in the following functions:
    1. `Market.getPNL()` should cache the state variables `priceFeed` and `indexToken`.

2. `Market.addPlugin()` should cache `plugins.length` to prevent loading `plugins` from storage on each iteration of the `for` loop. (**Update:** Fixed)
3. `Market.removePlugin()` should cache `plugins.length` to prevent loading `plugins` from storage on each iteration of the `for` loop.
4. `MarketRouter.getGlobalSize()` should cache `positionBooks.values().length` to prevent loading `positionBooks` from storage on each iteration of the `for` loop.
5. `MarketRouter.getAccountSize()` should cache `positionBooks.values().length` to prevent loading `positionBooks` from storage on each iteration of the `for` loop.
6. `ChainPriceFeed.getPrice()` should cache `sampleSpace`. (**Update:** Deprecated)
7. `FastPriceFeed._setPrice()` should cache `priceDataInterval`.

41. To improve readability in the `Market`, `OrderMgr`, `PositionAddMgr`, and `PositionSubMgr` contracts, remove the getter function `_valid()`, change the type of `marketValid` to `IMarketValid`, and replace all instances of `_valid()` with `marketValid`. (**Update:** Fixed)
42. Refactor the `if-else` statements in the function `MarketDataTypes.isValid()` for improved readability. (**Update:** Fixed)

```
if(!_params.isOpen && _params.isCreate) {
        if(_params._order.getFromOrder() > 0 ||
_params._order.extra2 > 0) {
                return false;
        }
}
return true;
```

43. Unlike a contract, library functions with `internal` visibility do not need to follow the leading underscore naming convention.
    1. `MarketLib._updateCumulativeFundingRate()` ⇒ `MarketLib.updateCumulativeFundingRate()`.
44. The check `require(_inputs.isValid(), "...")` in `MarketRouter.increasePosition()` is unnecessary since `_inputs.isValid()` checks that `_inputs._account` is not the zero address, but later in the function `_inputs._account` is overwritten and set to `msg.sender`. Remove the `require` statement. (**Update:** Fixed)
45. The following blocks of code in `MarketRouter.increasePosition()` and `MarketRouter._updateOrder()` have the same functionality as `TransferHelper.transferIn()`. Consider replacing the code blocks with `TransferHelper.transferIn()` for improved readability. (**Update:** Fixed)

```
// increasePosition()
IERC20(c).safeTransferFrom(
    msg.sender,
    _inputs._market,
    calculateEquivalentCollateralAmount(c,
_inputs.collateralDelta)
);
```

```
// _updateOrder()
IERC20(c).safeTransferFrom(
    msg.sender,
    _vars._market,
    calculateEquivalentCollateralAmount(c, _vars.pay())
);
```

46. Consider the need for the `onlyRole()` modifier used in `MarkerRouter.updatePositionBook()` since the caller is already validated with `require(markets.contains(msg.sender), "invalid market")`. (**Update:** Fixed)
47. In `MarketRouter.updatePositionBook()`, the following code block performs the same action of removing the position book twice. Since `positionBooks.remove()` does not revert when the given address is not in the `positionBooks` set, the only impact is increased gas costs when executing the function.

```
// (1) Get the position book address and remove from the
positionBooks set
positionBooks.remove(address(IMarket(_market).positionBo
ok()));

// (2) Get the position book address again and attempt
to remove from the set.
address _positionBook =
address(IMarket(_market).positionBook());
positionBooks.remove(_positionBook);
```

48. Contract state variables and functions with `internal` or `private` visibility should follow the leading underscore naming convention (i.e., `_functionName()` or `_stateVariableName`).
    1. `AcUpgradable.initBlock`. (**Update:** Fixed)
    2. `OrderMgr.getPrice()`. (**Update:** Fixed)
    3. `PositionAddMgr.commitIncreasePosition()`. (**Update:** Fixed)
    4. `PositionAddMgr.getPrice()`. (**Update:** Fixed)
    5. `PositionAddMgr.valLiq()`. (**Update:** Fixed)
    6. `OrderBook.setupTriggerAbove()`. (**Update:** Fixed)
49. Refactor the function `OrderMgr._shouldDeleteOrder()` to remove the `if` branch. (**Update:** Fixed)

```
function _shouldDeleteOrder(
    string memory errorMessage
) internal pure returns (bool) {
    return !StringsPlus.equals(errorMessage,
StringsPlus.POSITION_TRIGGER_ABOVE);
}
```

50. In `ChainPriceFeed`, use type `IPriceFeed` instead of `address` for the price feed contract to avoid unnecessary type casting throughout the contract.

(**Update:** Fixed)

```
mapping(address => IPriceFeed) public priceFeeds;
```

51. Functions with `public` visibility not referenced within the same contract should be changed to `external` visibility.
    1. `CoreVault.setLPFee()` . (**Update:** Fixed)
    2. `CoreVault.setCooldownDuration()` . (**Update:** Fixed)
52. The `private` function `FastPriceFeed._setPrice()` takes in an argument `_feed` , which is the contract address of the desired Chainlink price feed. All references to `_setPrice()` passes the state variable `chainPriceFeed` for the `_feed` argument. Consider removing the `_feed` argument and replacing all references with `chainPriceFeed` . (**Update:** Fixed)
53. The function `FastPriceFeed._setPriceData()` uses the incorrect operator during input validation. The arguments of `_setPriceData()` are each validated to be less than the respective maximum value. This is misleading since the function arguments should be validated to be less than **or equal** to the respective maximum value.
54. To simplify the deployment process of a new `Market` and eliminate the risk of initializing the `OrderBook` contract with incorrect `OrderStore` contracts, consider refactoring the `OrderBook` contract to deploy and `initialize()` the `OrderStore` contracts within its constructor.

```
// OrderBook.sol

constructor(address _f, bool _isLong) Ac(_f) {
        // deploy and initialize an OrderStore for open
orders
    OrderStore open = new OrderStore(_f);
    open.initialize(_isLong);    // deploy and
initialize an OrderStore for closed orders
OrderStore close = new OrderStore(_f);
close.initialize(_isLong);

isLong = _isLong;
openStore = IOrderStore(address(open));
closeStore = IOrderStore(address(close));
}
```

55. Refactor the calculations in the function `CoreVault.computationalCosts()` to improve readability. Consider using `CoreVault.getLPFee()` to obtain the corresponding fee amount. (**Update:** Fixed)

```
function computationalCosts(
    bool isBuy,
    uint256 amount
) public view override returns (uint256) {
        return (amount * getLPFee(isBuy)) /
```

```
FEE_RATE_PRECISION;
}
```

56. If an expression already returns a Boolean value, comparing that result to `true` or `false` is redundant, considering refactoring the following requirement checks for gas savings:
    1. `CoreVault._deposit()`: `require(false == isFreeze, "...")`.
57. Consider moving the state variable `vaultReward` in the `CoreVault` contract to the top, along with the other state variables, to improve readability.
58. Consider moving the event `LogIsFreeze` in the `CoreVault` contract to the top, along with the other events, to improve readability. (**Update:** Fixed)
59. Explicitly set the visibility of all contract state variables. The following contract state variables do not have an explicitly set visibility:
    1. `MarketConfigStruct.sol`
        1. `MAX_SLIPPAGE_BIT_POSITION`
        2. `MIN_LEV_BIT_POSITION`
        3. `MAX_LEV_BIT_POSITION`
        4. `MIN_PAY_BIT_POSITION`
        5. `MIN_COL_BIT_POSITION`
        6. `MAX_TRADE_AMOUNT_BIT_POSITION`
        7. `ALLOW_CLOSE_BIT_POSITION`
        8. `ALLOW_OPEN_BIT_POSITION`
        9. `DECIMALS_BIT_POSITION`
        10. `DECREASE_NUM_LIMIT_BIT_POSITION`
        11. `VALID_DECREASE_BIT_POSITION`
        12. `DENOMINATOR_SLIPPAGE`
60. Consider the need for the `MarketConfigStruct` library. The apparent goal for this library is to manually create a `struct` so that all fields are packed into a single storage slot. However, a custom implementation is not needed to achieve such a goal. In fact, the custom implementation increases the risk of potential bugs, deployment costs, and execution costs. Consider creating a `MarketConfig` struct and removing the `MarketConfigStruct` library.

```
struct MarketConfig {
        uint16 minSlippage;
        uint16 maxSlippage;
        uint16 minLev;
        uint16 maxLev;
        uint16 minPay;
        uint16 minCollateral;
        uint64 maxTradeAmount;
        bool allowClose;
        bool allowOpen;
        uint8 decimals;
        uint16 decreaseNumLimit;
        uint8 validDecrease;
}
```

61. Set contracts as `abstract` when they are not intended to be deployed directly but inherited by other contracts. Set the following contracts as `abstract`:

1. `Ac` . (**Update:** Fixed)
2. `AcUpgradable` . (**Update:** Fixed)
3. `MarketStorage` . (**Update:** Fixed)

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

- `f0f...352 ./contracts/vault/DeployDependency.sol`
- `7df...066 ./contracts/vault/ERC4626.sol`
- `91b...4bf ./contracts/vault/RewardDistributor.sol`
- `26c...287 ./contracts/vault/CoreVault.sol`
- `5e7...c34 ./contracts/vault/ERC20.sol`
- `507...eb5 ./contracts/vault/VaultRouter.sol`
- `a1e...571 ./contracts/vault/VaultReward.sol`
- `781...bef ./contracts/vault/interfaces/IVaultReward.sol`
- `211...30f ./contracts/vault/interfaces/ICoreVault.sol`
- `884...9c6 ./contracts/vault/interfaces/IVaultRouter.sol`
- `319...fb7 ./contracts/vault/interfaces/IRewardDistributor.sol`
- `7c5...787 ./contracts/referral/Referral.sol`
- `000...baa ./contracts/referral/interfaces/IReferral.sol`
- `141...5d5 ./contracts/fee/FeeRouter.sol`
- `846...d8a ./contracts/fee/FundFee.sol`
- `3cf...aa1 ./contracts/fee/FeeVault.sol`
- `dbe...04b ./contracts/fee/interfaces/IFeeRouter.sol`
- `fe5...c23 ./contracts/fee/interfaces/IFundFee.sol`
- `cb0...33a ./contracts/fee/interfaces/IFeeVault.sol`
- `b59...587 ./contracts/position/PositionStore.sol`
- `1f5...00b ./contracts/position/PositionBook.sol`
- `c94...f47 ./contracts/position/PositionStruct.sol`

- c00...0ac ./contracts/position/interfaces/IPositionStore.sol
- 014...834 ./contracts/position/interfaces/IPositionBook.sol
- 889...847 ./contracts/utils/Calc.sol
- 211...e1a ./contracts/utils/Strings.sol
- d69...e65 ./contracts/utils/TransferHelper.sol
- 362...894 ./contracts/utils/EnumerableValues.sol
- 419...08d ./contracts/openzeppelin/ERC721.sol
- 7df...066 ./contracts/openzeppelin/ERC4626.sol
- 5e7...c34 ./contracts/openzeppelin/ERC20.sol
- f76...1ff ./contracts/token/BaseToken.sol
- 34a...2e7 ./contracts/token/MintableBaseToken.sol
- 28a...432 ./contracts/token/BLE.sol
- b07...ce6 ./contracts/token/BPT.sol
- 9fe...db7 ./contracts/token/EsBLE.sol
- b1a...587 ./contracts/token/interfaces/IMintable.sol
- 1f0...6d4 ./contracts/mocker/ERC20Mocker.sol
- b0c...986 ./contracts/mocker/MockFeeStore.sol
- 7f9...074 ./contracts/mocker/MockVaultRouter.sol
- 0a4...735 ./contracts/mocker/MockFeeRouter.sol
- 4ea...c0c ./contracts/mocker/PositionBookMock.sol
- b55...6cf ./contracts/mocker/MockPositionMarket.sol
- 281...b94 ./contracts/mocker/MockChainPriceFeed.sol
- cee...1c5 ./contracts/mocker/MockMarket.sol
- 326...0a5 ./contracts/mocker/USDC.sol
- aa2...ec1 ./contracts/mocker/MockOracle.sol
- 9bc...233 ./contracts/stake/Booster.sol
- 44c...2d8 ./contracts/stake/RewardRouter.sol
- 767...0ee ./contracts/stake/Vester.sol
- 3e5...3fd ./contracts/stake/VaultRewardBase.sol
- 576...3bc ./contracts/stake/BLERouter.sol
- ac6...8c2 ./contracts/stake/BNFT.sol
- 33b...3c9 ./contracts/stake/TradeVolume.sol
- 8ce...386 ./contracts/stake/BLERewardTracker.sol
- d35...3f4 ./contracts/stake/EsBLETradeRewarder.sol
- a6b...0cc ./contracts/stake/interfaces/IBLERouter.sol
- b1a...587 ./contracts/stake/interfaces/IMintable.sol

- a52...4aa ./contracts/stake/interfaces/ITradeVolume.sol
- 4c9...d19 ./contracts/stake/interfaces/IVester.sol
- 99b...983 ./contracts/stake/interfaces/IBNFT.sol
- 22e...023 ./contracts/stake/interfaces/ITradePlugin.sol
- 116...e8a ./contracts/stake/interfaces/IRewardTracker.sol
- 96d...48e ./contracts/market/Market.sol
- 0da...281 ./contracts/market/GlobalDataTypes.sol
- 93d...4ab ./contracts/market/MarketConfigStruct.sol
- 8f5...866 ./contracts/market/MarketStorage.sol
- e7f...7b9 ./contracts/market/PositionSubMgr.sol
- 270...702 ./contracts/market/GlobalValid.sol
- 97f...567 ./contracts/market/MarketReader.sol
- 1d6...f79 ./contracts/market/MarketDataTypes.sol
- 630...f8f ./contracts/market/MarketLib.sol
- 3cf...7e5 ./contracts/market/MarketRouter.sol
- 952...afb ./contracts/market/MarketFactory.sol
- bc3...91e ./contracts/market/MarketValid.sol
- 7ed...7eb ./contracts/market/OrderMgr.sol
- 2cd...e21 ./contracts/market/PositionAddMgr.sol
- 46c...688 ./contracts/market/PostionSubMgrLib.sol
- c0b...d46 ./contracts/market/interfaces/IMarketValid.sol
- 0e0...1b1 ./contracts/market/interfaces/IMarketFactory.sol
- 487...8f5 ./contracts/market/interfaces/IMarketCallBackIntl.sol
- 89a...801 ./contracts/market/interfaces/IMarketRouter.sol
- 8ed...87d ./contracts/market/interfaces/IMarket.sol
- bf2...0d4 ./contracts/market/interfaces/IGlobalValid.sol
- b2c...dd0 ./contracts/market/interfaces/IMarketReader.sol
- 5a6...1ae ./contracts/ac/Ac.sol
- c9f...a65 ./contracts/ac/AcUpgradable.sol
- fde...fb3 ./contracts/order/OrderLib.sol
- cc5...b83 ./contracts/order/OrderStore.sol
- 86b...2e3 ./contracts/order/OrderBook.sol
- 620...105 ./contracts/order/OrderStruct.sol
- 591...cb9 ./contracts/order/interface/IOrderBook.sol
- 0d9...632 ./contracts/order/interface/IOrderStore.sol

- a20...277 ./contracts/oracle/Price.sol
- bf4...134 ./contracts/oracle/FastPriceFeed.sol
- 770...992 ./contracts/oracle/ChainPriceFeed.sol
- e38...76d ./contracts/oracle/interfaces/IPriceFeed.sol
- 3e3...ab1 ./contracts/oracle/interfaces/IChainPriceFeed.sol
- 0e7...ec7 ./contracts/oracle/interfaces/IPrice.sol
- ba9...0b3 ./contracts/oracle/interfaces/IGmxPriceFeed.sol
- 760...bdb ./contracts/oracle/interfaces/IFastPriceFeed.sol

Tests

- 0d9...bac ./test/utils.js
- bf3...d16 ./test/vault/prams.js
- cf9...ba0 ./test/vault/attack/vault.js
- 031...3f8 ./test/vault/user/sellDLP.js
- 30d...5fd ./test/vault/user/buyDLP.js
- 6d3...dcf ./test/vault/admin/deployVault.js
- 44d...8d6 ./test/testConver/vault/rewardDistributor/rewardDistributor.js
- 4e5...6ba ./test/testConver/vault/coreVault/coreVault.js
- 877...ec5 ./test/testConver/vault/vaultReward/vaultReward.js
- 2e7...89c ./test/testConver/vault/deployContract/deploy.js
- b2a...2b3 ./test/testConver/vault/vaultRouter/vaultRouter.js
- 3b6...754 ./test/testConver/referral/referral.js
- e61...ecc ./test/testConver/fee/fundVault/fundVault.js
- 197...23f ./test/testConver/fee/fundFee/fundFee.js
- f99...0c0 ./test/testConver/fee/feeRouter/feeRouter.js
- c36...33b ./test/testConver/position/positionBook.js
- 58c...fd7 ./test/testConver/position/positionStore.js
- 3da...851 ./test/testConver/market/marketRouter/marketRouter.js
- 6df...02c ./test/testConver/market/positionSubMgr/positionSubMgr.js
- 70c...509 ./test/testConver/market/orderMgr/orderMgr.js
- 5cf...6a5 ./test/testConver/market/market/market.js
- f16...eef ./test/testConver/market/positionAddMgr/positionAddMgr.js

- 7c1...f1a ./test/testConver/order/orderBook/OrderBook.js
- 4a1...4bb ./test/testConver/order/orderStore/OrderStore.js
- dfb...47d ./test/testService/vault/vaultRouter.js
- c55...dab ./test/testService/vault/vault/increasePosition.js
- e42...922 ./test/testService/vault/vaultRouter/increasePosition.js
- 8f1...d5c ./test/testService/utils/vaildPosition.js
- ee3...4ab ./test/testService/utils/getConfData.js
- cec...e10 ./test/testService/utils/vaildOrder.js
- 3be...96e ./test/testService/utils/buildParams.js
- a51...c01 ./test/testService/utils/fundingFee.js
- f8b...b2e ./test/testService/utils/utils.js
- a9f...dfc ./test/testService/utils/globalVaild.js
- 7dc...742 ./test/testService/utils/vaildData.js
- 06a...54a ./test/testService/utils/buildLogic.js
- e3c...49a ./test/testService/market/marketRouter/increaseOrder/increaseLimit.js
- fe8...d9b ./test/testService/market/marketRouter/cancelAll/cancelAll.js
- 5a1...3ea ./test/testService/market/marketRouter/cancelAll/cancelMutiUserPositon.js
- f7d...a34 ./test/testService/market/marketRouter/cancelAll/cancelMutiPosition.js
- 0c8...db4 ./test/testService/market/marketRouter/decreasePosition/vaildExample003.js
- 384...210 ./test/testService/market/marketRouter/decreasePosition/vaildRemoveCollGtFundingFee.js
- 503...719 ./test/testService/market/marketRouter/decreasePosition/vaildRemoveCollLtFundingFee.js
- 406...40e ./test/testService/market/marketRouter/decreasePosition/vaildExample001.js
- 845...754 ./test/testService/market/marketRouter/decreasePosition/vaild

PriceAndCollD.js

- e46...d0a ./test/testService/market/marketRouter/decreasePosition/vaildExample004.js
- 07a...b1b ./test/testService/market/marketRouter/decreasePosition/vaildDePosFeeAndReceive.js
- fa6...cab ./test/testService/market/marketRouter/decreasePosition/vaildExample005.js
- 174...d7b ./test/testService/market/marketRouter/decreasePosition/vaildExample002.js
- 057...8f3 ./test/testService/market/marketRouter/closePosition/vaildFundingFee.js
- 4fb...323 ./test/testService/market/marketRouter/closePosition/example001.js
- 1a1...038 ./test/testService/market/marketRouter/closePosition/closePosition.js
- cef...ae2 ./test/testService/market/marketRouter/decreaseTrigger/differUserTriggerTest.js
- 8d0...bb3 ./test/testService/market/marketRouter/decreaseTrigger/triggerLimitTest.js
- d41...eb4 ./test/testService/market/marketRouter/decreaseTrigger/maxTriggerTest.js
- 802...f9b ./test/testService/market/marketRouter/update/updateOrder.js
- bfe...d43 ./test/testService/market/marketRouter/increasePosition/vaildIncColLtFuningFee.js
- ac0...fc5 ./test/testService/market/marketRouter/increasePosition/validCollateral.js
- ac5...8c0 ./test/testService/market/marketRouter/increasePosition/vaildPnl.js

- a04...d9a ./test/testService/market/marketRouter/increasePosition/vaildTpAndSL.js
- e51...00b ./test/testService/market/marketRouter/increasePosition/vaildEvent.js
- 9f2...0dd ./test/testService/market/marketRouter/increasePosition/vaildVaultAndFees.js
- 48c...af2 ./test/testService/market/marketRouter/increasePosition/vaildAvgPrice.js
- 9e9...c50 ./test/testService/market/marketRouter/increasePosition/vaildIncCollGtFuningFee.js
- 74e...c08 ./test/testService/market/marketRouter/dePosShort/dePosShort.js
- 3ca...a46 ./test/testService/market/marketRouter/decreasePositionFromOrder/decreasePositionFromOrder.js
- 435...30e ./test/testService/market/marketRouter/aum/vaildpnl.js
- 4f7...1b7 ./test/testService/market/marketRouter/aum/validUserOpenPosLimit.js
- b7c...4eb ./test/testService/market/marketRouter/aum/vaildPnlUserOpenPosLimit.js
- e7f...4c0 ./test/testService/market/marketRouter/increasePositionFromOrder/increasePositionFromOrder.js
- 07c...a29 ./test/testService/market/market/getConf/getData.js
- 1a3...e9c ./test/testService/market/market/liqLong/example004.js
- 220...c21 ./test/testService/market/market/liqLong/example005.js
- 32b...789 ./test/testService/market/market/liqLong/example003.js
- 856...afe ./test/testService/market/market/liqLong/example001.js

- `004...5dc ./test/testService/market/market/liqLong/example002.js`
- `3db...838 ./test/testService/market/market/autoOpen/autoOpen.js`
- `3c0...a07 ./test/testService/market/market/autoOpen/example004.js`
- `581...311 ./test/testService/market/market/autoOpen/example003.js`
- `c01...b35 ./test/testService/market/market/autoOpen/example002.js`
- `72e...a73 ./test/testService/market/market/liqShort/partLiq.js`
- `00c...a55 ./test/testService/market/market/liqShort/LiqAll.js`
- `cbc...107 ./test/testService/market/market/liqShort/partLiq1.js`
- `ff7...002 ./test/testService/deploy/read.js`
- `670...2f3 ./test/testService/deploy/deployAllContract.js`
- `a9a...b20 ./test/testService/deploy/deploy.js`

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.8.3

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Automated Analysis

Slither

Unable to run slither.

Test Suite Results

The test suite contained 49 tests, all of which have passed.

Update: The Blex team added an additional test suite built with Foundry. The new test suite includes standard tests and fuzz tests as well. The new test suite had a total of `81` tests. However, `2` of these tests failed with the following output:

```
[FAIL. Reason: Expected an emit, but the call reverted
instead. Ensure you're testing the happy path when using
the `expectEmit` cheatcode.] testCV01001() (gas: 427838)
[FAIL. Reason: Expected an emit, but the call reverted
instead. Ensure you're testing the happy path when using
the `expectEmit` cheatcode.] testCV01002() (gas: 518889)
```

We recommend fixing the failed tests mentioned above. We cannot fully evaluate the quality of the tests since there are multiple test suites without a unified coverage report. However, we encourage the Blex team to continue improving their tests.

```
Test output omitted due to length.
```

Code Coverage

The tests provided were run successfully. However, the statement and branch coverages highly fluctuate with an average statement coverage of 59% and average branch coverage of 40%, where some contracts remain completely untested (`MarketConfigStruct.sol` , `MarketFactory.sol` , `MarketReader.sol` , `MarketValid.sol` , all three oracle contracts, `OrderLib.sol` , `Calc.sol` and `ERC*.sol`).

We highly recommend adding tests to cover all contracts and reach for 90% and higher branch- and statement-coverage.

Once the test suite has been improved to cover all the code, it can be further improved by using mutation testing.

Update: The Blex team added an additional test suite built with Foundry. Since there are multiple test suites, we were not able to obtain a unified coverage report.

File	% Stmts	% Branch	% Funcs	% Lines	Uncove Lines
ac/	55	16.67	63.64	62.96	
Ac.sol	100	100	100	100	
AcUpgradable.sol	52.63	16.67	60	61.54	... 9,95,96
fee/	89.55	54.63	92.31	87.57	
FeeRouter.sol	84.62	44.44	88.89	81.43	... 245,279
FeeVault.sol	100	50	100	100	
FundFee.sol	92.21	66	94.44	91.01	... 403,405
market/	47.9	28.7	54.24	49.71	
GlobalDataTypes.sol	100	100	100	100	
GlobalValid.sol	0	0	8.33	0	... 215,216
Market.sol	66.04	40.91	82.61	69.15	... 303,309
MarketConfigStruct.sol	0	0	0	0	... 229,234
MarketDataTypes.sol	50	30.77	77.78	51.28	... 172,173

File	% Stmts	% Branch	% Funcs	% Lines	Uncove Lines
MarketFactory.sol	0	0	16.67	0	... 163,168
MarketLib.sol	85.19	40.91	85.71	82.76	31,32,3 ,85
MarketReader.sol	0	0	0	0	... 285,28 7
MarketRouter.sol	83.33	42.5	95.24	87.8	... 499,50 1
MarketStorage.sol	100	100	100	100	
MarketValid.sol	0	0	0	0	... 451,45 5
OrderMgr.sol	56.14	32	77.78	47.44	... 309,31 2
PositionAddMgr.sol	91.8	55.56	100	94.94	48,96,1 37
PositionSubMgr.sol	87.01	54.17	84.62	87.63	... 308,31 7
PostionSubMgrLib.sol	64.29	57.69	100	69.57	... 47,62,6
oracle/	0	0	0	0	
ChainPriceFeed.sol	0	0	0	0	... 81,85,8

File	% Stmts	% Branch	% Funcs	% Lines	Uncove Lines
FastPriceFeed.sol	0	0	0	0	… 577,57 0
Price.sol	0	0	0	0	… 208,21 2
order/	84.35	47.22	80.85	87.01	
OrderBook.sol	83.64	46.67	80	85.88	… 216,21
OrderLib.sol	0	100	0	0	12
OrderStore.sol	100	54.55	100	100	
OrderStruct.sol	66.67	42.31	71.43	66.67	… 129,13
position/	85.19	57.76	90	86.17	
PositionBook.sol	83	57.61	88	83.8	… 443,44 7
PositionStore.sol	100	50	100	100	
PositionStruct.sol	81.25	64.29	80	85	41,70,7
referral/	65.12	36.36	81.25	71.93	
Referral.sol	65.12	36.36	81.25	71.93	… 224,23 4

File	% Stmts	% Branch	% Funcs	% Lines	Uncove Lines
utils/	51.35	31.25	53.33	50	
Calc.sol	0	0	0	0	... 24,31,3
Enumerable Values.sol	66.67	50	66.67	56.52	... 60,61,6
Strings.sol	50	100	50	50	19
TransferHel per.sol	66.67	50	83.33	72.73	93,95,9
vault/	58.66	41.84	54.63	59.59	
CoreVault.s ol	100	71.43	100	100	
DeployDep endency.sol	100	100	100	100	
ERC20.sol	0	0	0	0	... 366,37 1
ERC4626.s ol	0	0	0	0	... 320,32 9
RewardDist ributor.sol	90	50	85.71	92.59	47,85
VaultRewar d.sol	76.56	45.45	88.46	72.97	... 220,22 5
VaultRouter .sol	95.74	50	100	98.33	209

Changelog

- 2023-09-05 - Initial report
- 2023-10-30 - Final report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication or other making available of the report to you by Quantstamp.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites&aspo; owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that your access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information linked to, called by,

referenced by or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

 blex.io