CONSENSYS Diligence

MCDEX Mai Protocol V2 Audit

Date	May 2020
Lead Auditor	Martin Ortner
Co-auditors	Alexander Wade

1 Executive Summary

In May 2020, MCDEX asked us to conduct a security assessment of Mai Protocol V2, an extension of the Monte Carlo Decentralized Exchange platform (mcdex.io)

We performed this assessment over three calendar weeks: from May 18 to June 05, 2020.

1.1 Scope

Our review focused on the commit hash `4b198083ec4ae2d6851e101fc44ea333eaa3cd92`. A complete list of files in scope can be found in the Appendix.

1.2 Activity log

During the first week, our efforts were directed towards understanding the Mai Protocol V2 contract system, its interfaces, and how the various contracts and other entities interact with the system.

A kickoff meeting was held on May 18, 2020, after which a common communication channel was established. The assessment team used this channel to ask the client questions, as well as to communicate to the client any security-relevant issues as soon as they were found. The assessment team reviewed the provided documentation and began exploring the source code.

The assessment team noted to the client that:

- Inline code documentation is sparse
- The provided documentation was lacking a description about several interfaces and entities in the system

The mcdex.io team provided updates to the documentation during the assessment.

During the end-of-week progress meeting, the assessment team informed the client on the main focal points of the week and provided preliminary information about issues under investigation. Together with the client, it was established to set the assessment team's focus on `AMM`, `Perpetual`, and `Exchange` for the next week.

As the contract system in question was deployed on the mainnet and appeared to be live on the client's website, the assessment team reminded the client of the risks associated with making an unaudited system available to their users on mainnet and noted that the Perpetual contract held about 1300 ETH at the time. The client accepted this risk and stated that the risk is outlined with a banner on their website, stating:

> *ATTENTION：Audit is undergoing and this is Beta version. Trade at your own risk. The cap of collateral is $500k for Beta.*

During the second week, we continued diving deeper into `AMM`, `Perpetual`, and `Exchange`. To understand the system and its risks, we produced several ancillary visualizations (that can be seen throughout this report).

A high-level interface and actors diagram was shared with the client, and we requested they examine and verify that its layout was generally correct. We mainly requested this due to the sparse specification documents available at that time. The specification documents were updated or newly created during the assessment. The assessment team notified the client of several gaps and inconsistencies in the specification documents (e.g., general principles were not described like FundingLoss/SocialLoss).

During the third week, the assessment team focused on reviewing issues raised so far, grouping issues by general themes and providing recommendations, revisiting the specification flagging any inconsistencies, and preparing the report for the delivery on Friday.

2 Action Items

2.1 Reduce overall complexity

Complexity comes at the cost of security. Complex systems are harder to understand, harder to test, and harder to maintain.

For smart contract systems, the fault-intolerant environment of the EVM necessarily demands that security is the highest priority. Therefore, it should be a design goal of all smart contract systems to reduce complexity and make logic explicit wherever possible.

Mai V2 is a highly complex system:

- The contracts are continuously measuring the difference between the "mark price" of the `Perpetual` contract and Chainlink's ETH/USD index price. The percentage difference between these two values defines the "funding rate," which impacts the payouts of short and long positions in the contract.
    - As explained in the MCDEX docs, the calculation of accumulated funding payment per position is calculated using a 25-branch statement:
    - > *In order to calculate the `Acc`, consider that the funding rate will cross up to 4 special boundary values (`-GovMarkPremiumLimit`, `-GovFundingDampener`, `+GovFundingDampener`, `+GovMarkPremiumLimit`). 4 points segment the curve into 5 parts, so that the calculation can be arranged into 5 * 5 = 25 cases. In order to reduce the amount of calculation, the code is expanded into 25 branches.*
    - This calculation does not translate well into Solidity, requiring some ~200 lines of signed math operations to express the full range of options. (See `AMM.getAccumulatedFunding`).
    - Note, too, that `AMM.funding`, which can branch into `AMM.getAccumulatedFunding`, is called almost every time any of the Mai V2 contracts is interacted with. Many functions call `AMM.funding` multiple times. For example, `Exchange.matchOrders` can call `AMM.funding` up to `2 + (3 * makerOrderParams.length)` times.
- One contract, `Perpetual`, is split into two deployed instances: `Perpetual` itself, and `PerpetualProxy`, which routes all calls directly to `Perpetual` (using `CALL`, not `DELEGATECALL`). `Perpetual` has several functions that are access-restricted via the `onlyWhitelisted` modifier. Our understanding of the system is that two contracts should be whitelisted: `Exchange` and `PerpetualProxy`. `PerpetualProxy` implements its own access-control modifier, `onlyAMM`. The net result is that `AMM` calls `Perpetual` through `PerpetualProxy`, the `Exchange` calls `Perpetual` directly, and two separate access control mechanisms must function correctly for this to work as expected.
- Throughout the contracts, a common theme is the use of both signed and unsigned math, as well as math dealing with "wad"-denominated values versus "raw" values. Because variables are not named in a way that suggests they are either "signed" or "unsigned," "wad" or "raw," reading the Mai V2 contracts often requires a lot of backtracking to variable declarations.
- Several inconsistencies in method and variable naming add to the confusion. For example, the modifier `AMM.onlyBroker` seems to suggest that the caller should be the "broker." However, the modifier actually

checks that `perpetualProxy.currentBroker(msg.sender) == authorizedBroker()`, which is another way of saying that "`PerpetualProxy` needs to be the caller's broker."

Recommendation:

Reducing overall complexity is no simple task, and addressing this system's complexity will require careful thought and consideration outside of the scope of this review. In general, prioritize the following concepts:

- **Optimize for readability.** Ensure that code is as easy to understand as possible. Implement clear and consistent naming conventions, group similar functions within the same file, and generally attempt to structure and organize the code so that humans can read and understand it best.
- **Reduce function side effects.** Rather than include `funding` (or its `Perpetual` counterpart, `markPrice`) as an implicit call in every function, refactor the code to have each `public` or `external` function call `funding` only once.
    - Additionally, calls to `funding` should be explicit. As an example, consider `Perpetual.isSafe`. The name implies that the function is a "getter," which should make some simple check and return a value. Instead, `isSafe` is a state-changing function that can possibly branch into the impossible-to-follow math of `funding`: `Perpetual.isSafe -> Perpetual.markPrice -> AMM.currentMarkPrice -> AMM.funding`. As a result, even simple concepts like `isSafe` become incredibly difficult to understand.

Related:

Recommendations	Priority
Implement clear, consistent naming conventions for all contracts	High
Clarify confusing use of signed integers	High
Refactor `PerpetualProxy`	High
Use individually typed setter methods instead of a combined `set*Prameter` method	Medium
Prefix variables that are expected to denominated in "wads" to make them distinguishable from integers	Medium
Import 3rd party libraries from their original source and keep them unchanged instead of copying their content into a new library	Low
Avoid redefining the same structs	Low
Methods should be declared external	Low
Issues	**Severity**
Perpetual - Variable shadowing in constructor	Minor
Where possible, a specific contract type should be used rather than `address`	Minor

2.2 Increase the overall quality and quantity of testing

Several findings of this assessment suggest that Mai V2 is inadequately tested:

- Issue 6.1 showed that a critical feature, order cancellation, did not function whatsoever.
    - The function in question (`cancelOrder`) seems to behave as expected: an order's "trader" or "broker" can call `Exchange.cancelOrder` , adding the hash of the order in question to the `Exchange.cancelled` mapping. However, none of `Exchange` 's trading functions check that submitted orders are in this mapping, so cancelled orders can be processed all the same.
    - Although 2 unit tests check the behavior of the function in question (`cancelOrder`), no tests check whether a "cancelled" order can still be traded in the `Exchange` . This suggests that more care should be taken to test behavior across multiple functions, rather than merely testing functions in isolation.
- Issue 6.5 describes an incorrectly-set function visibility. The function (`liquidateFrom`) was marked `public` , rather than `internal` . This oversight allows anyone to force liquidated positions on other users and attempt liquidations at improper times.
    - While it would be strange to test whether a function had a correctly-set visibility, the mistake implies that insufficient consideration has been given to `liquidate` 's internal behavior. Proper testing requires careful consideration of the various branches execution can take and requires a familiarity with the code that should have spotted this.

Recommendation:

Implementing a robust, complete test suite requires careful consideration outside of the scope of this review. In general, prioritize the following concepts:

- **Write tests that encapsulate the specification.** Tests should address each of a system's requirements. A system's requirements should be clearly defined within the system design specification. Ensure that the Mai V2 test suite accurately reflects the most up-to-date specification and includes checks for all of the requirements mentioned therein.
- **Perform extensive fuzz-testing on mathematical functions.** Mai V2's monolithic funding rate calculation (and other formulas) introduce severe dependencies on the mathematical approximations present in `LibMath` , the proper use of these approximations, and a staggeringly-wide range of values that can be assigned to global parameters via admin functions. To ensure these work together under any condition, the system should be tested using a wide range of invalid, unexpected, or random data.

Related:

Issues	Severity
Exchange - CancelOrder has no effect	Critical
Perpetual - `liquidateFrom` should not have `public` visibility	Major
Perpetual - withdrawFromInsuranceFund should check `wadAmount` instead of `rawAmount`	Major
Perpetual - `withdraw` should only be available in `NORMAL` state	Major
AMM - `funding` can be called in emergency mode	Major
LibMathSigned - `wpowi` returns an invalid result for a negative exponent	Medium

2.3 Address codebase fragility

Software is considered "fragile" when issues or changes in one part of the system can have side-effects in conceptually unrelated parts of the codebase. Fragile software tends to break easily and may be challenging to maintain.

Our assessment uncovered several indicators of software fragility in Mai V2:

- Issue 6.8 describes that liquidity providers can never be sure of the result of calls to `addLiquidity` and `removeLiquidity`. The amount of collateral received for burned shares, and the number of shares received for provided collateral is based on the system's current price and total shares in circulation. These values can fluctuate significantly for many reasons:
  - Oracle price updates may introduce a new price to the system. Significant deviations from expected values may result in unexpected gains or losses for users.
  - Frontrunning by other users (whether on purpose or not) will affect the current price and total share amount.
  - Adjustments to global variable configuration by the system admin do not come with a delay, so changes will directly impact users' subsequent actions.
- System configuration by administrators primarily occurs in `Perpetual` (via inherited `PerpetualGovernance`) and `AMM` (via inherited `AMMGovernance`). Both configuration features are accessed through monolithic `setGovernanceParameter` functions, where an input `bytes32 key` is compared against all existing parameter names for a match. If a match is found, the parameter is set to the input `int256 value`.
  - If future development adds or removes configurable parameters, the change will have a broad impact on the entire configuration system.
  - `int256 value` is not a sufficiently-descriptive value for many configurable parameters. Many parameters must first convert this to a `uint` via `LibMathSigned.toUint256`, which rejects negative input values. As a result, if a parameter is introduced that requires a high enough `uint` value, these functions will not work as the positive values of `int256` do not go higher than `2 ** 255 - 1`.
  - By using a multipurpose function like `setGovernanceParameter`, configurable parameters are not afforded the type safety checks Solidity would provide if standard, single-purpose setter methods were used.

Recommendation:

Building an anti-fragile system requires careful thought and consideration outside of the scope of this review. In general, prioritize the following concepts:

- **Follow the single-responsibility principle of functions.** This principle states that functions should have responsibility for a single part of the system's functionality and that their purpose should be narrowly-aligned with that responsibility. Avoid functions that "do everything" (like `setGovernanceParameter`), and avoid functions that touch every other function (like `funding` and `markPrice`).
- **Reduce reliance on external systems.** Whether the "external system" refers to the Chainlink oracle or admin control, the contracts should avoid blindly and immediately consuming and conforming to the

arbitrary inputs of external systems. External systems can introduce significant change at a moment's notice: the oracle may wildly impact the index price, and admins may suddenly make large adjustments to fee rates, lot sizes, premiums, and other critically-important values. When reducing reliance on external systems, make sure users can interact with the system in a consistent, expected manner.

Related:

Recommendations	Priority
Use individually typed setter methods instead of a combined `set*Prameter` method	Medium
Avoid redefining the same structs	Low
Issues	**Severity**
Unpredictable behavior due to front running or general bad timing	Major
Exchange - validateOrderParam does not check against `SUPPORTED_ORDER_VERSION`	Medium
Signed data may be usable cross-chain	Medium
Oracle - Unchecked oracle response timestamp and integer over/underflow	Medium
AMM - Liquidity provider may lose up to `lotSize` when removing liquidity	Medium
Exchange - insufficient input validation in `matchOrders`	Medium
AMM - Amount of collateral spent or shares received may be unpredictable for liquidity provider	Medium
AMM - Unchecked return value in `ShareToken.mint`	Minor

2.4 Improve documentation and create a complete technical specification

A system's design specification and supporting documentation should be almost as important as the system's implementation itself.

Security assessments depend on a complete technical specification to understand *how a system is supposed to function*. When a behavior is not specified (or is specified incorrectly), security assessments must base their knowledge in assumptions, leading to less effective review.

Maintaining and updating code relies on proper supporting documentation to know *why the system is implemented in a specific way*. If code maintainers cannot reference documentation, they must rely on memory or assistance to make high-quality changes.

Our assessment notes several problems with Mai V2 documentation:

- Inline commenting is sparse to non-existent.
- Provided documentation lacks a description of some interfaces and entities in the system.
- Some documentation is out-of-date and refers to outdated concepts and terms.

Related:

Recommendations	Priority
Improve documentation and provide a complete specification	High
`LibTypes.Status.SETTLING` should be renamed to `LibTypes.Status.EMERGENCY`	Medium
Issues	**Severity**
Unpredictable behavior due to front running or general bad timing	Major

2.5 Ensure system states, roles, and permissions are sufficiently restrictive

Smart contract code should strive to be *strict*. Strict code behaves predictably, is easier to maintain, and increases a system's ability to handle nonideal conditions.

Our assessment of the Mai V2 protocol found that many of its states, roles, and permissions are loosely defined:

- Mai V2's administrator role assigns complete control over most elements of the protocol to a single account. This control includes setting individual account balances, draining the system's insurance fund, changing system addresses and permissions, and more (See Actors for a more detailed description).
    - The extent to which administrator permissions can impact the contracts suggests that future plans to transition the administrator role to a DAO model have not been well thought through. In its current configuration, it would be incredibly difficult to transition the management of the administrator's extensive permissions to a smart contract.
    - If the administrator key is compromised, an attacker will have complete and instant access to the underlying assets held within the contracts.
    - If the administrator key is somehow destroyed or lost, the contracts will be unable to enter the global "EMERGENCY" mode.
- Both `AMM` and `Perpetual` make use of OpenZeppelin's `WhitelistedRole` module, which includes two roles: "Whitelisted" and "WhitelistAdmin." In `AMM`, the Whitelisted role is assumed to be the `Exchange` contract only. In `Perpetual`, the Whitelisted role is assumed to be both `PerpetualProxy` and `Exchange`. As described in "Refactor `PerpetualProxy`," the use of `WhitelistedRole` in `Perpetual` has significant downsides:
    - From the perspective of a user or external reviewer, it is much harder to determine which entities should be able to perform which actions.
    - Because `PerpetualProxy` and `Exchange` are both Whitelisted, they have equivalent permissions in `Perpetual`. If vulnerabilities are discovered in either contract that allow arbitrary calls to `Perpetual`, the Whitelisted role's permissions will allow the `Exchange` to act like `PerpetualProxy`, and vice versa. Additionally, the method `WhitelistedRole.renounceWhitelisted` would enable such a vulnerability to completely break large portions of the system.
    - Future updates to the system may introduce additional contracts to the Whitelisted role. It may be challenging to ensure that new contracts do not introduce vulnerabilities due to their Whitelisted

permission. Additionally, if old contracts are no longer used, the Whitelisted role necessitates that the WhitelistAdmin remember to remove their permissions.

- Mai V2 has three primary states: `NORMAL`, `SETTLING` (aka `EMERGENCY`), and `SETTLED`.
  - Issue 6.13 describes that there is no restriction on the duration of the `SETTLING` stage. Once activated, the admin can choose whether the stage lasts minutes, days, or years.
  - Issue 6.22 describes that the `SETTLING` stage can be entered multiple times before the `SETTLED` stage is reached. In effect, this allows the system `settlementPrice` to be set multiple times, making it difficult for users to count on any specific outcome for the liquidation process.
  - Some functions can be called during improper contract states, as described in issue 6.2 and issue 6.3.

Recommendation:

- **Follow the Principle of Least Privilege.** Ensure that each role within the system is given *only* the bare minimum permissions to perform their responsibilities.
- **Document the use of administrator permissions.** For users to know what they can expect from Mai V2, the administrator's roles and responsibilities should be clearly and completely documented and communicated.
- **Monitor the usage of administrator permissions.** To ensure the administrator key's potential compromise is detected, monitor transactions and events in Mai V2 for administrator action.

Related:

Recommendations	Priority
Refactor `PerpetualProxy`	High
Issues	**Severity**
AMM - `funding` can be called in emergency mode	Major
Perpetual - `withdraw` should only be available in `NORMAL` state	Major
Perpetual - `liquidateFrom` should not have `public` visibility	Major
Unpredictable behavior due to front running or general bad timing	Major
Perpetual - Administrators can put the system into emergency mode indefinitely	Medium
Perpetual - `beginGlobalSettlement` can be called multiple times	Minor

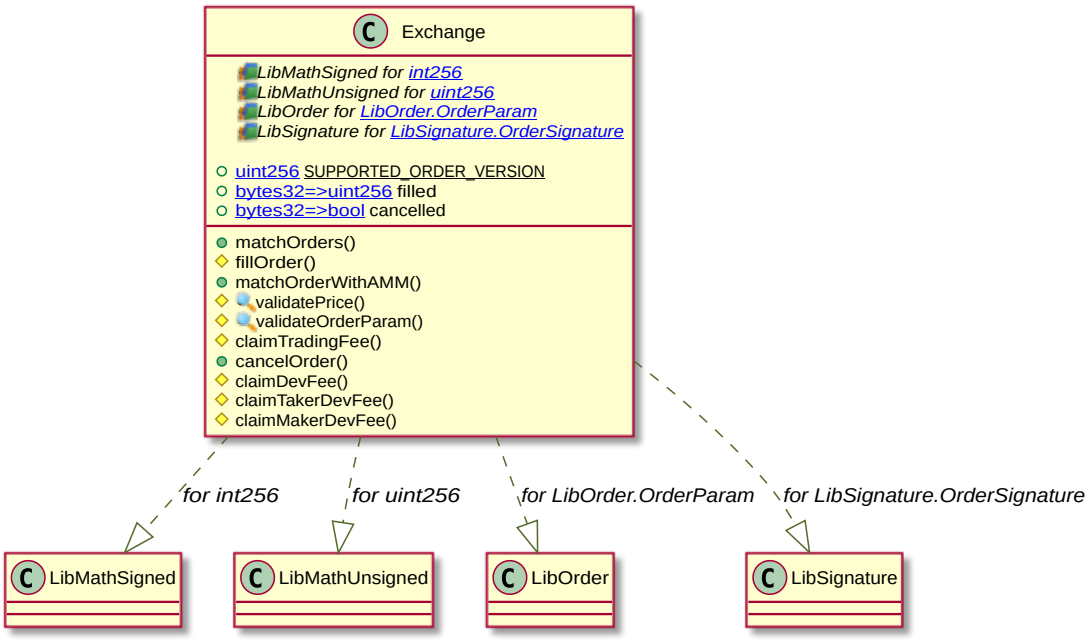
3 System Overview

The mcdex.io Mai Protocol V2 aims to create decentralized Perpetual contracts on the Ethereum blockchain. Users can either trade with the on-chain automated market maker (AMM) or the off-chain order book (Exchange). The system accepts `ETH` or any `ERC20` compliant token (with at max. 18 decimals) as collateral.

The system under review (documentation) consists of the following components, with the main parts being the `Exchange`, `AMM`, and `Perpetual`. It is initially deployed in `NORMAL` operating mode and can be set to `EMERGENCY` or `SETTLED` state by an administrative account at any time.

Exchange

Provides interfaces for off-chain order book trading. Brokers can match signed orders from traders. A taker can only match with either Exchange or AMM.

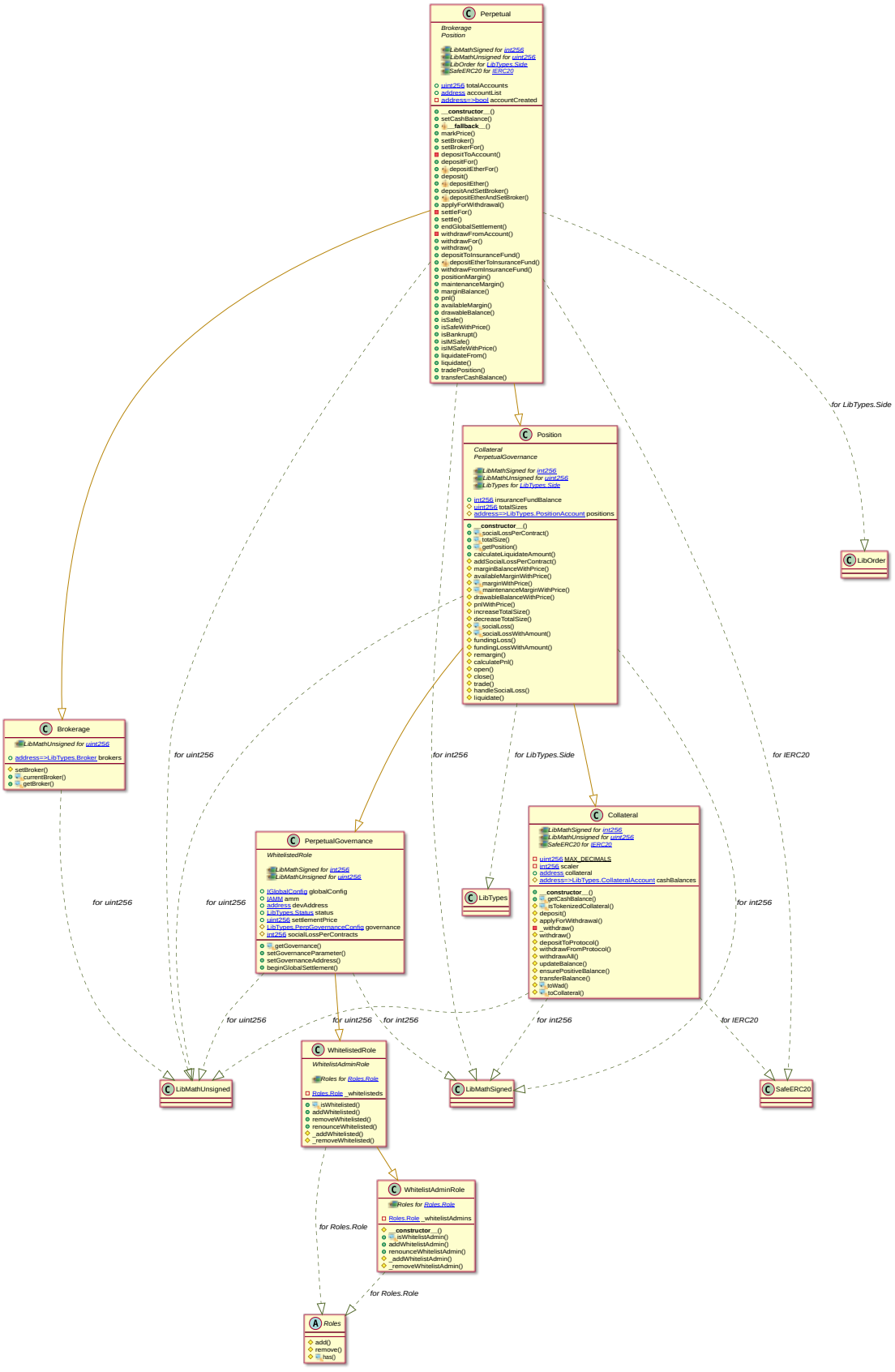


Exchange.sol

Perpetual

Holds assets owned by users and provides interfaces to manipulate balance and position. One perpetual contract is serving one trading pair. Traders have to deposit collateral in `ETH` or the configured `ERC20` token before interacting with the `Exchange` or `AMM`. Balances are only updated in the margin accounts when executing trades. Collateral token/`ETH` transfers are only executed when withdrawing or depositing funds. The collateral token or `ETH` is specified when deploying the token and cannot be changed. Special care should be taken when deploying a token with zero decimals as its calculations might be subject to rounding errors.

Perpetual is the main contract of the system that - for example - specifies the current `AMM`, `GlobalConfig` addresses being used as well as allows an administrator to put the contract into emergency mode.

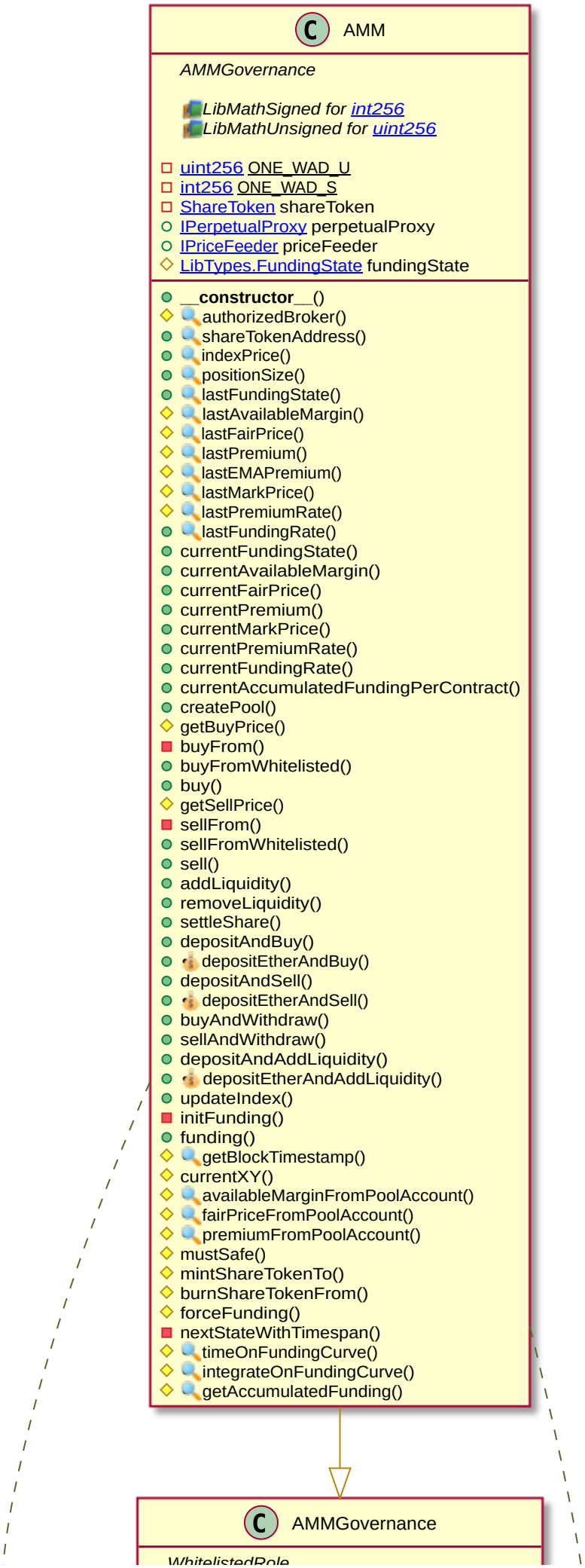


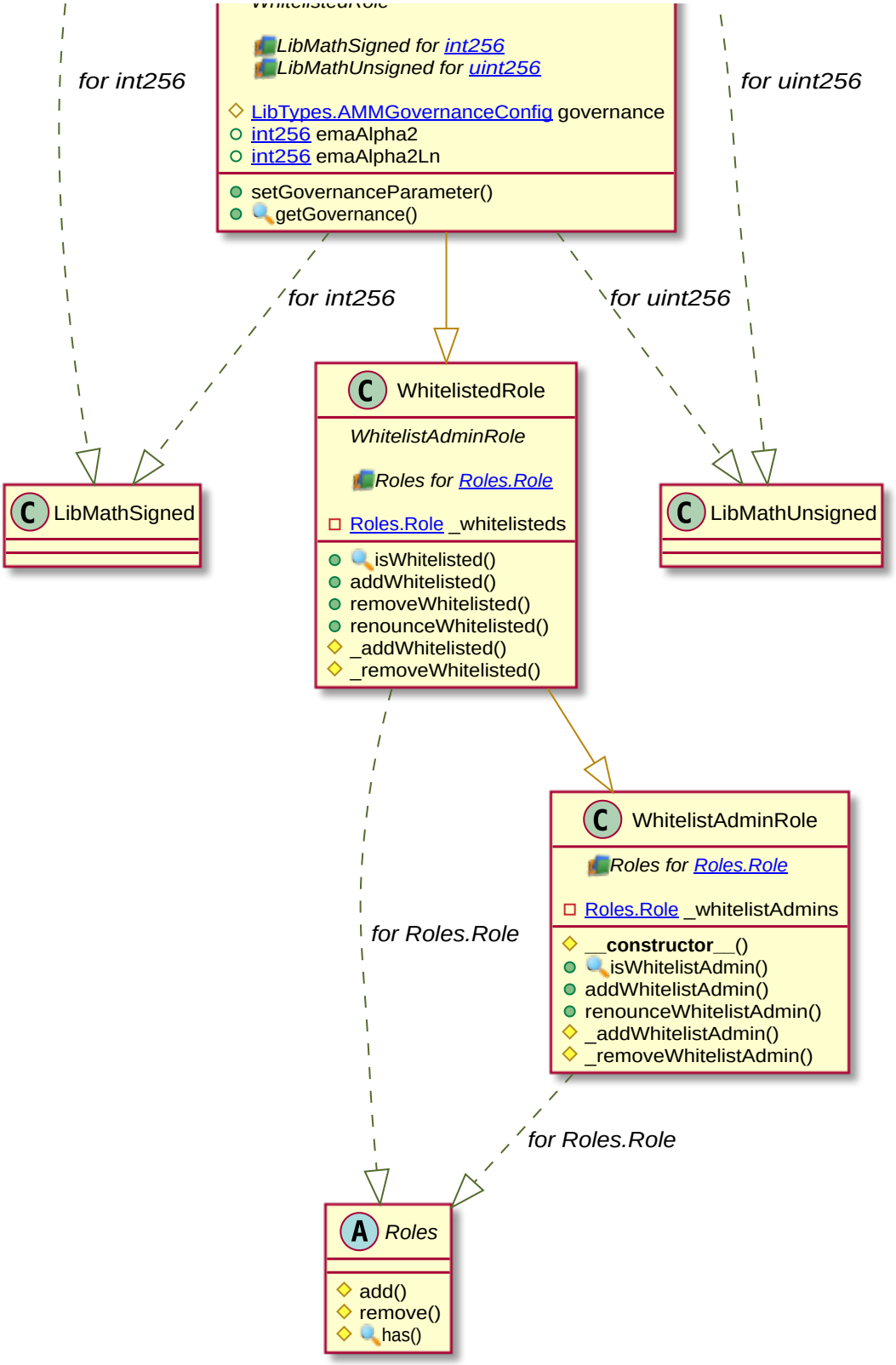
Perpetual.sol

AMM

The automated market maker provides functionality for trading, funding rate calculation, and liquidity management that burns and mints `ShareToken` that represent a liquidity providers' share of the pool.

`Perpetual` defines the current `AMM` contract address that is being used and, therefore, `Perpetual` can upgrade to a new `AMM` by setting a new `AMM` contract address.

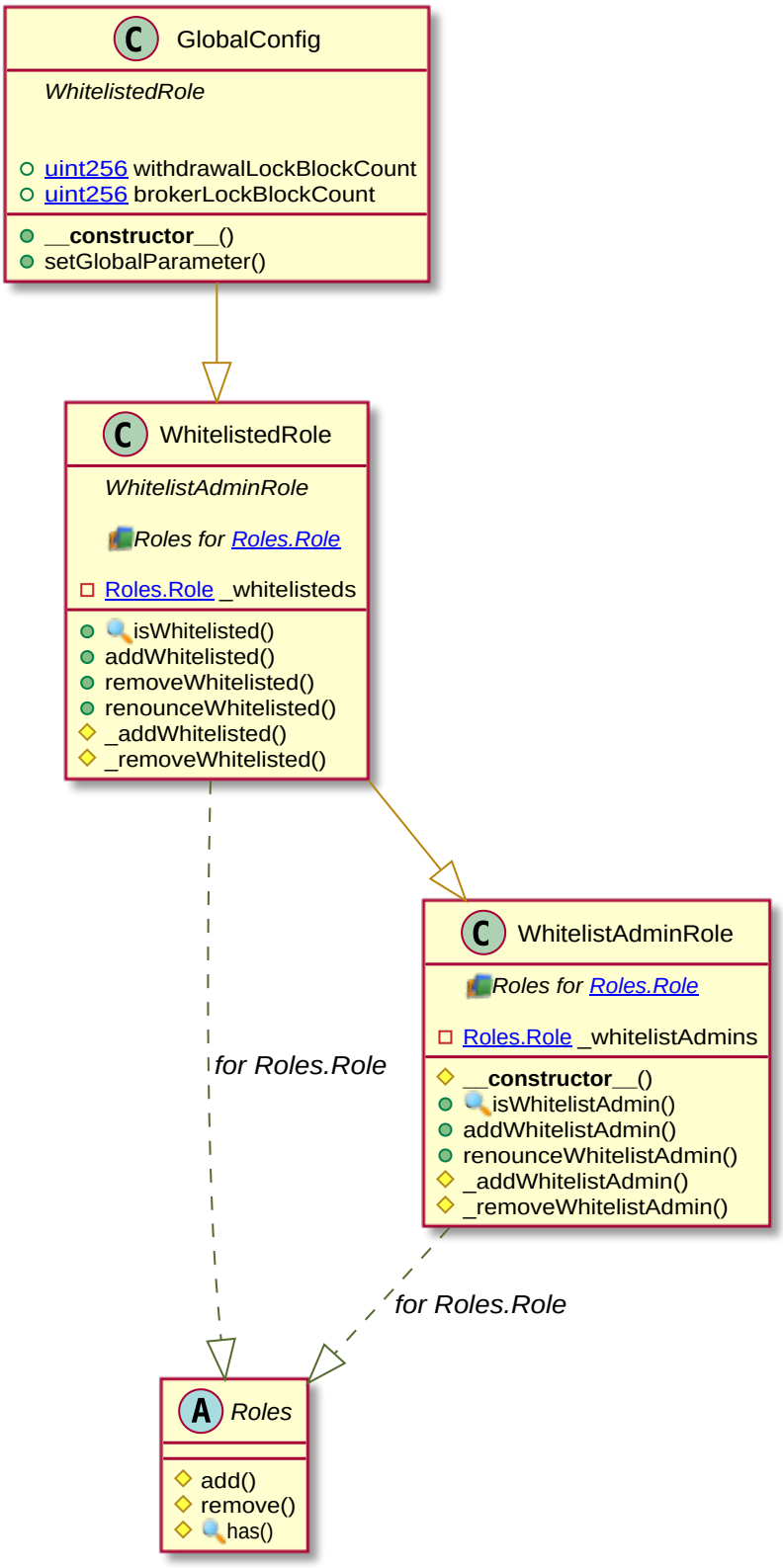

AMM
AMMGovernance
LibMathSigned for int256
LibMathUnsigned for uint256
uint256 ONE_WAD_U
int256 ONE_WAD_S
ShareToken shareToken
IPerpetualProxy perpetualProxy
IPriceFeeder priceFeeder
LibTypes.FundingState fundingState
__constructor__()
authorizedBroker()
shareTokenAddress()
indexPrice()
positionSize()
lastFundingState()
lastAvailableMargin()
lastFairPrice()
lastPremium()
lastEMAPremium()
lastMarkPrice()
lastPremiumRate()
lastFundingRate()
currentFundingState()
currentAvailableMargin()
currentFairPrice()
currentPremium()
currentMarkPrice()
currentPremiumRate()
currentFundingRate()
currentAccumulatedFundingPerContract()
createPool()
getBuyPrice()
buyFrom()
buyFromWhitelisted()
buy()
getSellPrice()
sellFrom()
sellFromWhitelisted()
sell()
addLiquidity()
removeLiquidity()
settleShare()
depositAndBuy()
depositEtherAndBuy()
depositAndSell()
depositEtherAndSell()
buyAndWithdraw()
sellAndWithdraw()
depositAndAddLiquidity()
depositEtherAndAddLiquidity()
updateIndex()
initFunding()
funding()
getBlockTimestamp()
currentXY()
availableMarginFromPoolAccount()
fairPriceFromPoolAccount()
premiumFromPoolAccount()
mustSafe()
mintShareTokenTo()
burnShareTokenFrom()
forceFunding()
nextStateWithTimespan()
timeOnFundingCurve()
integrateOnFundingCurve()
getAccumulatedFunding()
AMMGovernance
WhitelistedRole



AMM.sol

Global Config

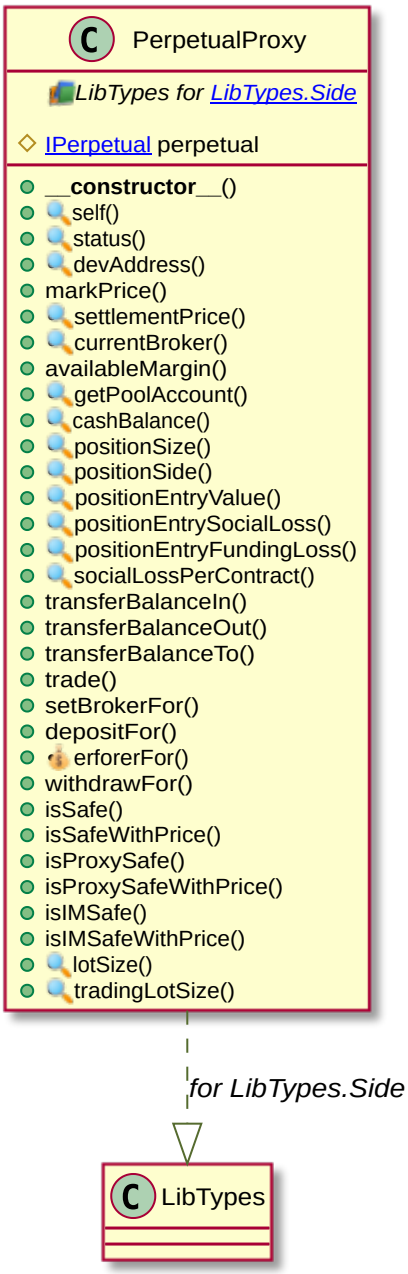
Stores global system parameters. Currently only used to store and set the block delay for `withdrawal` and `broker` updates.



GlobalConfig.sol

Perpetual Proxy

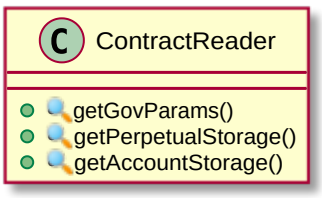
This contract is a workaround to be able to upgrade the AMM and ensure it has a constant address.



PerpetualProxy.sol

Contract Reader

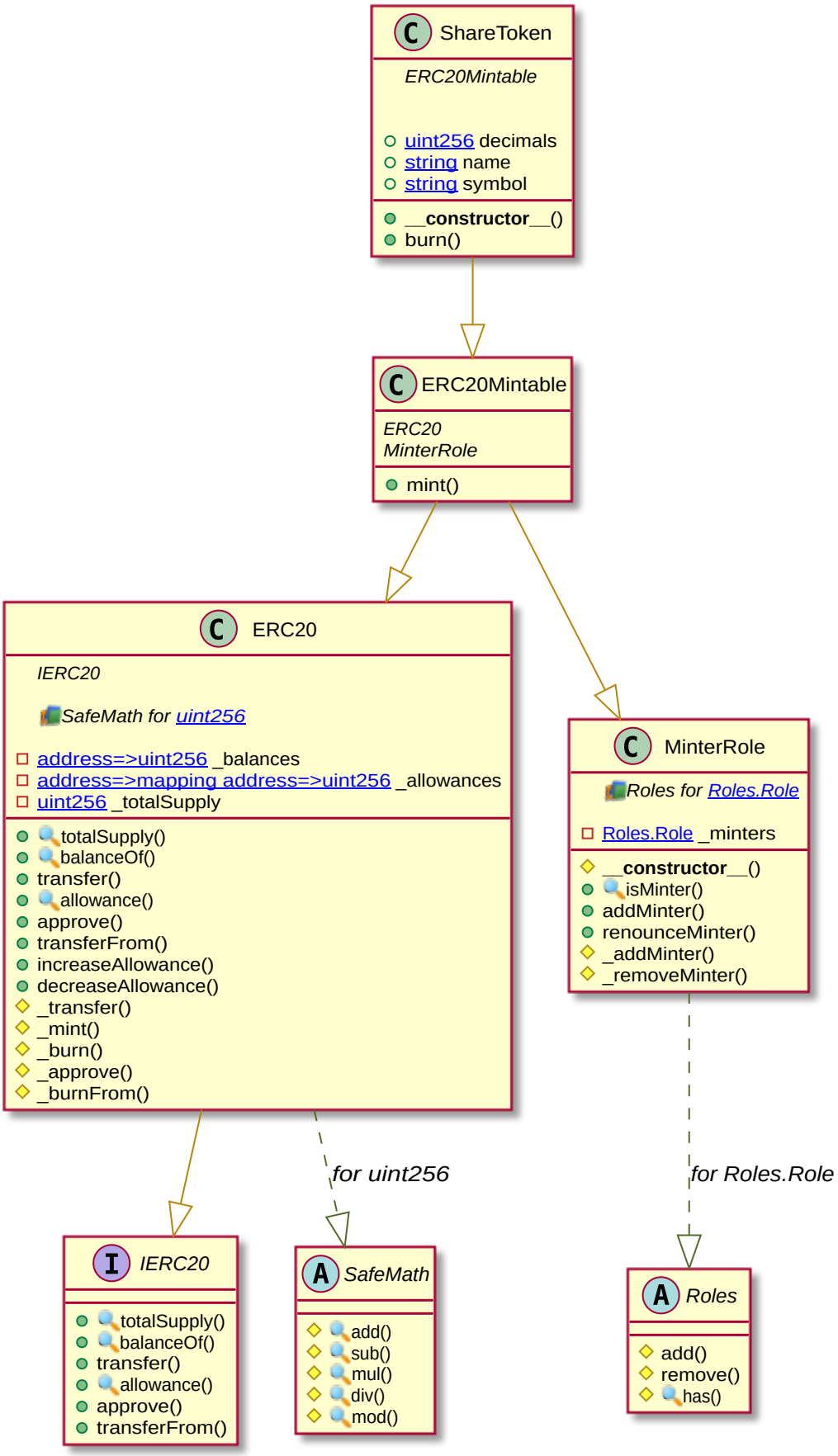
An auxiliary contract to read state and data from the system. This contract is not used by any other contract in the system.



ContractReader.sol

ERC20 Token (Customized): ShareToken

A customized `ERC20` token initially owned by the deployer that allows `MinterRole` to `burn` and `mint` tokens. The `ShareToken` is minted to liquidity providers according to their share of the pool.



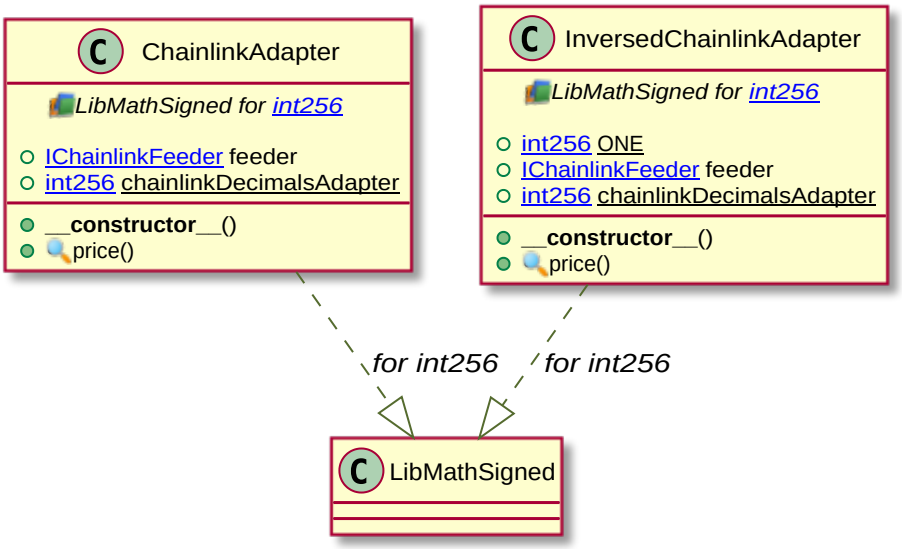
ShareToken.sol

ERC20 Token (Standard): Collateral

An `ERC20` standard token following the `@openzeppelin/contracts/token/ERC20/IERC20.sol` interface description used as collateral for the perpetual contract.

Oracle (External): Reversed/-ChainlinkAdapter

Chainlink oracle adapter used by `AMM` to retrieve the index price.



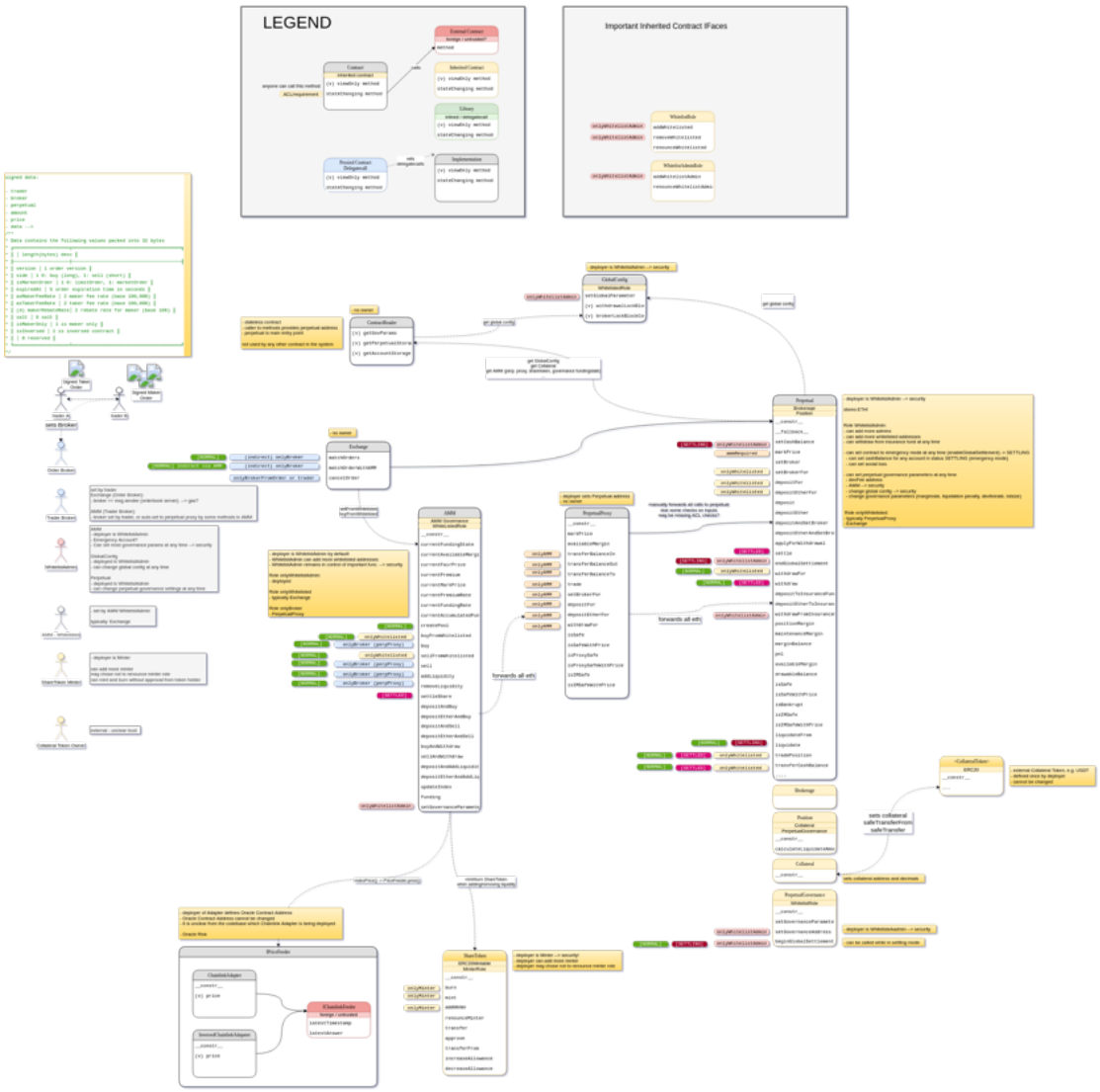
ChainlinkAdapter.sol and InverseChainlinkAdapter.sol

4 Security Specification

This section describes, **from a security perspective**, the expected behavior of the system under review. It is not a substitute for documentation. The purpose of this section is to outline trust relationships and describe specific security properties that were identified by the assessment team.

The contract system can be in one of three states:

- `NORMAL` (default)
- `SETTLING` hereby also referred to as `EMERGENCY` mode
- `SETTLED`



Contract System Actors, Interfaces and Access Control

4.1 Actors

Actors are listed below with a general description of their role in the system followed by more details on their respective abilities for specific components:

- `deployer`
  - deploys a contract in the system

    - may take the role of an `administrator`
- `administrator`
    - may change system or global parameters
    - may switch-out components (upgrading)
    - may put the contract into `EMERGENCY` or `SETTLED` mode
    - may perform the global settlement in case the contract is put into `EMERGENCY` mode
    - may choose to keep the system in `EMERGENCY` mode without settling
    - may manipulate balances of users in `EMERGENCY` mode
    - may withdraw from the insurance fund at any time
    - may hold special permissions in system tokens (`Sharetoken` : `mint` , `burn`)
- `trader`
    - must first deposit collateral
    - signs orders for the off-chain `Exchange`
    - delegates to a `broker` for matching off-chain orders with `Exchange`
    - delegates to a `broker` for the on-chain `AMM`
- `broker`
    - set by a trader
    - matches orders on behalf of the `trader`
- `oracle`
    - an external ChainLink price feed
    - oracle answers must be trusted by the system
    - price slippage may occur
    - oracle may fail to provide recent prices or there may be a gap to the real price (DoS, targeted attacks, exploited trust to oracle owner)
    - oracle may provide wrong prices
    - oracle may cease to exist
- `liquidity provider`
    - provides liquidity in the form of collateral to the `AMM`
    - Gets `ShareToken` minted in return
- `ShareToken Holder`
    - an account with a non-zero balance of `ShareToken` aka. an active `liquidity provider`
- `Collateral Token Holder`
    - an account with a non-zero balance of the configured collateral token (`ERC20` or `ETH`)
- `anyone`
    - any other account on the blockchain may interact with the contract system without taking a specific role

4.2 Trust Model

Exchange

- owner: none, standalone contract
- Tracks filled and canceled orders
- Verifies order signatures.

    - Defines allowed signed order version
- Typically called by a trader's broker
- Caller provides the address of the perpetual contract used when matching
    - Exchange retrieves system parameters (e.g. lotsize) and performs trades
- Order signature includes perpetual's address, trader, broker, and trading data

Actors

- `anyone`
    - can see canceled/filled orders
- `trader`
    - submits order to off-line order book
    - must set `broker` for trades
    - can specify positive or negative fee's (either broker or trader pays)
    - can cancel orders
- `broker`
    - main actor for the contract. matches orders on behalf of traders
        - orders can only be matched if `msg.sender` is set as `broker` for affected orders
    - `trader` can also be `broker`
    - can cancel orders

Perpetual

- owner: `deployer`, `administrator`
- accepts `ETH`
- explicitly rejects `ETH` via fallback function
- governed by one or more `administrators` with initial `administrator` being `the deployer`
- defines critical perpetual parameters that immediately affect all users
    - the address of the `AMM`
    - the address of `GlobalConfig`
    - the system status (e.g. `NORMAL`, `EMERGENCY`, `SETTLED`)
    - the settlement price
    - margin rate, liquidation penalty, fee rates, lot sizes, socia loss
- uses openzeppelin `WhitelistedRole`
- only minimal initial configuration is enforced in the constructor (globalConfig), there is a risk that variables might stay uninitialized and therefore operating out of specification
- users must verify configuration before interacting with the system
    - `administrators` can set parameters or "switch-out" components (`AMM`) at any time (admin front-running opportunity)
    - `administrators` might add more `administrators`

Actors

- `deployer`
    - is `administrator`
    - deploying address (individual) may choose not to renounce the administrative role

- `administrator`
  - can change perpetual parameters at any time (front-running opportunity)
  - can "switch-out" `AMM` and `GlobalConfig` at any time (front-running opportunity)
  - can add other `WhitelistAdmin`'s
  - can renounce `administrator` role
  - can add more `whitelisted` addresses (typically `PerpetualProxy` and `Exchange`)
  - may choose to front-run own or other transactions changing perpetual system parameters
  - can withdraw from insurance fund at any time
  - can put contract into `EMERGENCY` mode at any time
  - may choose to stay in `EMERGENCY` mode indefinitely
  - may put contract into `EMERGENCY` mode even when in `EMERGENCY` mode
  - can manipulate cash balances of any account in `EMERGENCY` mode
  - can set social loss
- account holder (`trader`, `broker`)
  - can apply for withdrawal (delayed by configurable amount of blocks)
  - can withdraw from account (when not in `EMERGENCY` mode)
  - can change their own broker
  - liquidate their own account
  - can call settle after `EMERGENCY` mode has ended
- `whitelisted` (typically `perpetual proxy` for `AMM` or `Exchange` directly)
  - can do anything `anyone` can
  - can trade positions for any account
  - can transfer cash balances for any account
  - can withdraw for any account (when in `NORMAL` mode)
  - can deposit for any account
  - can set broker for any account
- `anyone`
  - can deposit to open an account
  - can set their own broker
  - liquidate any account (see issue)
  - can mark price
  - deposit to insurance fund
  - check if an account is safe
  - read account information

AMM

- owner: `deployer`, `administrator`
- accepts `ETH` and forwards it to `PerpetualProxy` which in turn forwards it to `Perpetual`
- governed by one or more `administrators` with initial `administrator` being the `deployer`
- defines critical AMM parameters that immediately affect all users
  - mathematical factors (funding dampener)
  - premium size and limits

    - fee rates
- reads configuration from `perpetual` via `perpetualProxy`
- uses openzeppelin `WhitelistedRole`
- only minimal initial configuration is enforced in the constructor (perpetual proxy, pricefeed, and ShareToken address)
    - there is a risk that variables might stay uninitialized and therefore operating out of specification
    - perpetual proxy, pricefeed, and ShareToken address cannot be changed
- users must verify configuration before interacting with the system
    - `administrators` can set parameters at any time (admin front-running opportunity)
    - `administrators` might add more `administrators`
- trading operations are only allowed in `NORMAL` state

Actors

- `deployer`
    - is `administrator`
    - deploying address (individual) may choose not to renounce administrative role
- `administrator`
    - can change `AMM` parameters at any time (front-running opportunity)
    - can add other `WhitelistAdmin`'s
    - can renounce `administrator` role
    - can add more `whitelisted` addresses (typically `Exchange`)
    - may choose to front-run own or other transactions changing AMM system parameters
- account holder (`trader`)
    - can deposit collateral
    - can withdraw collateral
    - can implicitly set their broker to `PerpetualProxy` when using compound functions like `depositAndBuy`
- `trader` with `broker` set to `PerpetualProxy`
    - can buy/sell
    - can create a pool (only one pool)
    - can add liquidity to pool (only if pool has been created)
    - can buy/sell
- `liquidity provider` is a `trader` with `broker` set to `PerpetualProxy`
    - remove liquidity
    - settleShare after `EMERGENCY` mode has ended
- `whitelisted` (typically `Exchange`)
    - can sell from any account
    - can buy for any account
- `anyone`
    - can deposit to open an account (which also sets broker to `PerpetualProxy`)
    - update the index price
    - read current contract information

GlobalConfig

- owner: `deployer`, `administrator`
- governed by one or more `administrators` with initial `administrator` being the `deployer`
- defines critical global parameters
- uses openzeppelin `WhitelistedRole` but only makes use of `WhitelistAdmin`
  - may consider using `Ownable` instead
- can set `withdrawalLockBlockCount` and `brokerLockBlockCount` to arbitrary values
  - can disable the block delays completely due to missing input validation
  - values are initially set to zero which is unsafe
- initial configuration is not enforced in the constructor, there is a risk that variables might stay uninitialized and therefore operating out of specification
- users must verify configuration before interacting with the system
  - `administrators` can set parameters at any time (admin front-running opportunity)

Actors

- `deployer`
  - is `administrator`
  - deploying address (individual) may choose not to renounce administrative role
- `administrator`
  - can change global parameters at any time (front-running opportunity)
  - can add other `WhitelistAdmin`'s
  - can renounce `administrator` role
  - may choose to front-run own or other transactions changing global system parameters
- `anyone`
  - can read the settings

PerpetualProxy

- owner: none, standalone contract
- serves as constant account address of AMM to perpetual
- does not store state by itself
- retrieves `AMM` address from `perpetual` configuration
- restricts most access to `AMM` address

Actors

- `deployer`
  - provides address of `Perpetual` on deployment (cannot be changed)
- `AMM` configured in `Perpetual`
  - can transfer balances
  - can trade
  - can set broker for any accounts
  - can deposit for any account
  - can withdraw for any account

- `anyone`
  - can mark price if `AMM` is set in `Perpetual`
  - can read contract information like availableMargin
  - can check if account is safe

ContractReader

- owner: none, standalone contract
- view only, does not store any state
- external interface, not used by any other contract in the system
- caller to method provides address to `Perpetual`

Actors

- `anyone`
  - can interface with the contract to read GovParams, PerpetualStorage, and AccountStorage

ShareToken (Custom ERC20 Token)

- owner: `deployer`, `minter (administrator)`
- `minter` role is in full control of the token
- there can be multiple `minter` accounts

Actors

- `deployer`
  - is `minter (administrator)`
  - deploying address (individual) may choose not to renounce administrative role
- `minter (administrator)`
  - can `mint` an arbitrary amount of tokens to any address
  - can `burn` an arbitrary amount of tokens without the holders approval
  - can nominate other `minter`'s
  - can renounce own `minter` role
  - cannot renounce other `minter`'s role
- `ShareToken Holder`
  - can interact with the token interface in accordance with the `ERC20` specification (`transfer`)
  - can transfer token to an account that is unknown to the `AMM`
- `anyone`
  - can interact with the token interface in accordance with the `ERC20` specification

Collateral (Standard ERC20 Token)

- external ERC20 Token
- external token must be audited before accepting it as collateral for the system
- external token might be broken (wrong interfaces, implementation)
- external token might call back into `Perpetual` directly (re-entrancy)

- external token might implement callbacks and allow affected accounts (`from` , `to` addresses) to re-enter `Perpetual` before and after token transfers (beware of `ERC777`) (re-entrancy)
- external token is configured when deploying `Perpetual` . Make sure token decimals of the token reflect the decimals configured when deploying `Perpetual` .

Actors

- `ShareToken Holder`
  - can interact with the token interface in accordance with the `ERC20` specification
  - can provide token as liquidity to `AMM` to be `liquidity provider`
- `anyone`
  - can interact with the token interface in accordance with the `ERC20` specification

Oracle (*ChainlinkAdapter)

- owner: none, standalone contract
- used by `AMM` to fetch the index price
- Users must understand the inherent risks of oracles

Actors

- `anyone`
  - can retrieve price information from the oracle

5 Recommendations

5.1 Refactor `PerpetualProxy`

Description

`PerpetualProxy` is a forwarding contract that mirrors the `Perpetual` interface and provides a few wrappers for `Perpetual` functions. While the inclusion of the word `Proxy` implies that `PerpetualProxy` is a standard `delegatecall` proxy, its operations all use `call`. This means that `Perpetual` holds the state used by `PerpetualProxy`, with `PerpetualProxy`'s only state being a single address pointing to `Perpetual`.

While there is evidence within MCDEX's docs that `PerpetualProxy` originally held additional state, this is no longer the case. However, `PerpetualProxy` does still implement additional access-control logic that may be a holdover from a previous version. Namely, it includes the `onlyAMM` modifier, which restricts function call access to the `AMM` contract. Because `PerpetualProxy` does not hold state, this modifier must query `Perpetual`:

code/contracts/proxy/PerpetualProxy.sol:L13-L18

```
IPerpetual perpetual;

modifier onlyAMM() {
    require(msg.sender == address(perpetual.amm()), "invalid caller");
    _;
}
```

Note that `PerpetualProxy` is not an abstract calldata forwarder; it does not include a fallback function that forwards `msg.data` to `Perpetual`. Rather, each of the functions `PerpetualProxy` needs to call are a part of its own interface. Many of these functions are restricted to the `amm` contract, by use of the `onlyAMM` modifier:

code/contracts/proxy/PerpetualProxy.sol:L110-L124

```
function setBrokerFor(address guy, address broker) public onlyAMM {
    perpetual.setBrokerFor(guy, broker);
}

function depositFor(address guy, uint256 amount) public onlyAMM {
    perpetual.depositFor(guy, amount);
}

function depositEtherFor(address guy) public payable onlyAMM {
    perpetual.depositEtherFor.value(msg.value)(guy);
}

function withdrawFor(address payable guy, uint256 amount) public onlyAMM {
    perpetual.withdrawFor(guy, amount);
}
```

Of course, `Perpetual` and `PerpetualProxy` do not share state, and `Perpetual`'s functions can be called directly. This separation of logic and state has resulted in two separate access control implementations: the `onlyAMM` modifier in `PerpetualProxy`, and an `onlyWhitelisted` modifier in `Perpetual`. In order to successfully restrict a function's access to the `AMM` contract only, both modifiers must be employed:

- `PerpetualProxy` ensures the caller is the `AMM` contract via `onlyAMM`, then forwards the call to `Perpetual`:

code/contracts/proxy/PerpetualProxy.sol:L110-L112

```
function setBrokerFor(address guy, address broker) public onlyAMM {
    perpetual.setBrokerFor(guy, broker);
}
```

- `Perpetual` then needs to check that the caller (`PerpetualProxy`) is whitelisted via `onlyWhitelisted`:

code/contracts/perpetual/Perpetual.sol:L64-L66

```
function setBrokerFor(address guy, address broker) public onlyWhitelisted {
    setBroker(guy, broker, globalConfig.brokerLockBlockCount());
}
```

The `onlyWhitelisted` modifier, which is pulled from OpenZeppelin's `WhitelistedRole` contract, allows multiple whitelisted addresses. In the case of `Perpetual`, `PerpetualProxy` is not the only whitelisted address: the `Exchange` contract is also whitelisted. Additionally, the whitelist admin role in OpenZeppelin's `WhitelistAdminRole` contract allows additional whitelisted addresses to be added, each of which would have the same permissions as `Exchange` and `PerpetualProxy`.

Conclusion

The two-contract system is complicated, which is compounded by the use of the whitelist access control system. Technically, `Exchange` has the same access to `Perpetual` as the `AMM` contract, and vice-versa. Should issues be found or introduced in either `Exchange` or `AMM` that allow for arbitrary external calls, several components of the system may break or be tampered with. Further additions to the list of whitelisted addresses or whitelisted admins may have similar consequences.

Recommendation

- Remove `PerpetualProxy` entirely, as it no longer serves a purpose
- Implement the `onlyAMM` modifier within `Perpetual`, and replace `onlyWhitelisted` in `Perpetual` with `onlyAMM` where applicable
- Remove `onlyWhitelisted` in `Perpetual`, and implement an `onlyExchange` modifier where applicable
- Review system roles and permissions, and ensure that each contract is only given the minimum level of access needed to function effectively

5.2 Clarify confusing use of signed integers

Description

One factor that significantly introduces complexity to the smart contract system is the excessive use of signed integers for convenience and to encode implicit logic.

In practice, this degrades code readability, auditability, and security as it breaks assumptions humans might have formed based on the variable's use or name.

Examples

- variables declared as signed int that must not be negative: e.g., `insuranceFundBalance`

The variable is declared as a signed integer. However, the value of the insurance fund should never be allowed to be negative. In fact, it cannot be negative unless someone withdraws more funds than available. Since it is a signed integer it can theoretically be negative (permanently or for a short amount of time potentially during reentrant calls). To counter that, special care must be taken and explicit checks are added while simply falling back to declaring the variable `uint` would have avoided the necessity of adding more complexity.

code/contracts/perpetual/Position.sol:L15-L15

```
int256 public insuranceFundBalance;
```

code/contracts/perpetual/Perpetual.sol:L192-L202

```
function depositEtherToInsuranceFund() public payable {
    require(!isTokenizedCollateral(), "ether not acceptable");
    require(msg.value > 0, "invalid amount");

    int256 wadAmount = depositToProtocol(msg.sender, msg.value);
    insuranceFundBalance = insuranceFundBalance.add(wadAmount);

    require(insuranceFundBalance >= 0, "negtive insurance fund");

    emit UpdateInsuranceFund(insuranceFundBalance);
}
```

- multiple declarations of the same const for different signedness

code/contracts/liquidity/AMM.sol:L17-L18

```
uint256 private constant ONE_WAD_U = 10**18;
int256 private constant ONE_WAD_S = 10**18;
```

- implicit logic based on the signedness of an integer: `fee`

If the `fee` is positive, the amount is sent from `guy -> devAddress`. If the `fee` is negative, the amount is sent from `devAddress -> guy`.

Let alone, that this mechanism shifts responsibility to verify the sanity of system and order parameters to the entity matching orders (usually the broker) as the signed integer order fee rate defines if the broker or the trader pays fees.

code/contracts/exchange/Exchange.sol:L200-L212

```
int256 hard = price.wmul(openedAmount).toInt256().wmul(feeRate);
int256 soft = price.wmul(closedAmount).toInt256().wmul(feeRate);
int256 fee = hard.add(soft);
address devAddress = perpetual.devAddress();
if (fee > 0) {
    int256 available = perpetual.availableMargin(guy);
    require(available >= hard, "dev margin");
    fee = fee.min(available);
    perpetual.transferCashBalance(guy, devAddress, fee.toUint256());
} else if (fee < 0) {
    perpetual.transferCashBalance(devAddress, guy, fee.neg().toUint256());
    require(perpetual.isSafe(devAddress), "dev unsafe");
}
```

code/contracts/exchange/Exchange.sol:L93-L95

```
// trading fee
int256 takerTradingFee =
amount.wmul(price).toInt256().wmul(takerOrderParam.takerFeeRate());
claimTradingFee(perpetual, takerOrderParam.trader, takerTradingFee);
```

- `transferBalance` can never be called with a negative value but `wadAmount` is signed int.

code/contracts/perpetual/Collateral.sol:L140-L144

```
function transferBalance(address from, address to, int256 wadAmount) internal {
    if (wadAmount == 0) {
        return;
    }
    require(wadAmount > 0, "bug: invalid transfer amount");
```

that's also why explicit conversions to `int256` are required:

code/contracts/perpetual/Perpetual.sol:L313-L316

```
function transferCashBalance(address from, address to, uint256 amount) public
onlyWhitelisted {
    require(status != LibTypes.Status.SETTLING, "wrong perpetual status");
    transferBalance(from, to, amount.toInt256());
}
```

Recommendation

Rework the smart contract system design and declare signed integers only where they are absolutely needed. Refrain from declaring signed integers out of convenience when used with arithmetical operations. Refrain from encoding logic - like the direction of funds flow - into the signedness of the value and make it explicit instead.

Clearly explain why variables are signed and reflect the type of arguments and statevars used in the method's docstring. The more explicit the code is and the less complex it is, the easier it is to verify security assumptions.

5.3 Improve documentation and provide a complete specification

Description

Mai V2 lacks inline code documentation describing the purpose and relationships of source-units, their contracts, methods, and variables. Additionally, supporting documentation is frequently out-of-date, and many interfaces, roles, states, and permissions are missing entirely.

Recommendation

- Rather than duplicating function description in external documentation, provide inline documentation using Solidity's natspec format, as this will be easier to maintain.
- Provide supporting inline comments for critical functionality:
  - Outline why certain values are used and what purpose they serve.
  - Describe acceptable ranges for inputs, outputs, and intermediary calculations.
  - Elaborate on security concerns for critical methods and make your developers or external reviewers aware of any functions that require special attention due to their risk profile in the system.
- Improve, update, and complete the Mai V2 specification:
  - Ensure it is up-to-date at all times and implement the logic as specified without any deviations (e.g. deviation between Solidity math implementation and specification pseudocode).
  - Include a security discussion in the specification and inform users, developers, and reviewers of the risks attached to the system or components that require special attention.

Examples

The following non-exhaustive list provides an overview of various inconsistencies encountered during review. We highly recommend reviewing all documentation for accuracy and completeness as additional issues are likely to exist.

Inconsistent, unclear or insufficient explanation

- `Perpetual` Wrong state requirements that have since been corrected

mcdexio/documents@ `f5c1bd7`

- `Perpetual` Wrong state requirement `NORMAL` for `withdraw` while the code checks for `!SETTLING` which resolves to `NORMAL` and `SETTLED`

https://github.com/mcdexio/documents/blob/0a44d7ec48e09e2d229a3c5b77501235d4de82b3/en/perpetual-interfaces.md

- `Perpetual` inaccurate function signature

`totalSize(Side side)` should be `totalSize(LibTypes.Side side)` (and multiple other occurrences). Keep the function signature and types as accurately updated with the codebase.

https://github.com/mcdexio/documents/blob/0a44d7ec48e09e2d229a3c5b77501235d4de82b3/en/perpetual-interfaces.md

- `AMM` internal specification inconsistencies
  - Missing sources for mathematical calculations
  - Most of the Variable and Method names do not reflect actual names in code: `GovPoolFeeRate`, `PoolAvailableMargin`, and others.
  - `createPool` does not mention that it can only be called once otherwise `initFunding` bails
  - `createPool` does not mention the state requirement `NORMAL`
  - `buyFromPool` does not exist and should be `buyFrom`. It is also missing the state requirement `NORMAL`.
  - `buyFromPool` inconsistent requirement `BlockTime < DeadLine` which actually is `require(getBlockTimestamp() <= deadline, "deadline exceeded");` in code.
  - `buyFromPool` does not specify a minimum amount.
  - `buyFromPool` states `The trader buy/long. Can be called by anyone.` while it can only be called if the caller set the broker to `perpetualProxy` (which is mentioned as a confusing requirement `broker == LiquidityPool`).
  - `buyFromPool` does not mention the lotsize
  - `sellFromPool` similar inconsistencies to `buyFromPool`
  - `AddLiquidity` does not mention he state `NORMAL` as a requirement.
  - `AddLiquidity` unclear statement `The unit of "Amount" is contract.`
  - `RemoveLiqudity` does not mention that up to a lotsize of balance might be lost
  - `UpdateIndex` does not mention that the caller might be awarded a premium
  - `funding` states `isEmergency` as a requirement which is not state.
  - `funding` does not check the state requirement and can be called at any time.
  - `funding` steps are inconsistent with the code. E.g. when `lastFundingTime==0` `forceFunding()` just returns and does not set the `LastFundingTime` to `BlockTime`.
  - `funding` duplicate definition and deviating specification of formulas (even though they are the same): `v0 = LastEMAPremium; vt = (LastEMAPremium - LastPremium) * Pow(GovEMAAlpha2, n) + LastPremium` vs. `- v0 = LastEMAPremium; vt = (LastEMAPremium - LastPremium) * Pow(1 - GovEMAAlpha, n) + LastPremium` (here)

https://github.com/mcdexio/documents/blob/b94b98a806d29d7ce135e1011b094868e07eeb5d/en/internal-amm.md#createpoolamount

- `depositToInsuranceFund` unclear who would deposit to an insurance fund that can be drained by admins at any time.
- `LibTypes.Side` should add a description for when and how `FLAT` is used.

https://github.com/mcdexio/documents/blob/0a44d7ec48e09e2d229a3c5b77501235d4de82b3/en/perpetual-interfaces.md

- `Exchange` vague requirement for amounts array

> *Length of parameter 'amounts' should equal to the length of 'makerOrderParams'.*

https://github.com/mcdexio/documents/blob/0a44d7ec48e09e2d229a3c5b77501235d4de82b3/en/perpetual-interfaces.md

- `Perpetual` admin functionality vague state requirements

> *Perpetual.beginGlobalSettlement: Enter the "Emergency" status with a "settlement price". In this status, all trades and withdrawals will be disabled until "endGlobalSettlement"*

Unclear specification. `SETTLING` is referred to as `EMERGENCY` mode but it is not mentioned here. Stick to one distinct state description and use it throughout the specification and in code.

> *Perpetual.setCashBalance: Modify account.cashBalance. Can only be called in the "global settlement" status*

Inconsistent and unclear use of state name `global settlement`. This should state that this method can only be used in `EMERGENCY` or `SETTLING` mode. Stick to one name and use it throughout the specification and in code.

> *Perpetual.endGlobalSettlement: Enter the "global settlement" status. In this status, all traders can withdraw their MarginBalance*

Unclear what state is being entered right now. This should state that `SETTLED` mode is entered.

> *Perpetual.withdrawFromInsuranceFund: Withdraw collateral from insurance fund. Typically happen in the "global settlement" status*

Vague description of when this is allowed to be used when it basically can be called by an admin at any time as there is no state requirement.

https://github.com/mcdexio/documents/blob/0a44d7ec48e09e2d229a3c5b77501235d4de82b3/en/perpetual-admin-functions.md

- Unclear if it is by design that parameters can be changed by an admin at any time (including upgrading the system)

https://github.com/mcdexio/documents/blob/0a44d7ec48e09e2d229a3c5b77501235d4de82b3/en/perpetual-admin-functions.md

- `GlobalConfig` specification does not mention that there can be multiple admins, admins can add other admins, and whitelist accounts. Whitelisted accounts are not used with this contract.

https://github.com/mcdexio/documents/blob/0a44d7ec48e09e2d229a3c5b77501235d4de82b3/en/perpetual-admin-functions.md

- Contracts that provide admin functionality should clearly state who is assigned admin powers initially (deployed), whether the deployed keeps its admin role or renounces it, what other accounts get roles assigned (admin and whitelisted) to allow uses to audit a specific setup of the system. Note that processes need to be in place to manage privileged accounts (e.g. remove admins when they are compromised, remove privileges when they are no longer used or components are being upgraded)

https://github.com/mcdexio/documents/blob/0a44d7ec48e09e2d229a3c5b77501235d4de82b3/en/perpetual-admin-functions.md

- `AMM` describe the liquidity provider user journey.

Create pool must be called first and can only be called once. A pool cannot be created for an empty amount. A pool must exist for others to provide liquidity. A pool is not created automatically if none exists. Liquidity providers cannot provide zero amount.

The function descriptions should outline the requirements to call these methods more clearly. E.g. `buy, sell, addLiquidity, removeLiquidity` can only be called if the caller set the broker to `PerpetualProxy`.

- Clearly define and explain the operating values and boundaries for configuration parameters

https://github.com/mcdexio/documents/blob/b94b98a806d29d7ce135e1011b094868e07eeb5d/en/internal-amm.md#governance

- `Perpetual` outlines three states `Normal`, `Emergency` and `GlobalSettled` but the implementation refers to these states as `Normal`, `Settling` and `Settled`.

https://github.com/mcdexio/documents/blob/b94b98a806d29d7ce135e1011b094868e07eeb5d/en/internal-perpetual.md

- `Perpetual` methods state requirement `isEmergency==FALSE` while the implementation checks `status==NORMAL`.

https://github.com/mcdexio/documents/blob/b94b98a806d29d7ce135e1011b094868e07eeb5d/en/internal-perpetual.md

- `Perpetual` implements no method `buy` / `sell` (`AMM` does)

https://github.com/mcdexio/documents/blob/b94b98a806d29d7ce135e1011b094868e07eeb5d/en/internal-perpetual.md

- `Perpetual.liquidate` does not state that method can only be called in `status.NORMAL` or `status.SETTLING`

https://github.com/mcdexio/documents/blob/b94b98a806d29d7ce135e1011b094868e07eeb5d/en/internal-perpetual.md

Initially Missing but since then updated

- A description for `socialLoss` and `fundingLoss` was not present and was added towards the 2nd half of the audit.

mcdexio/documents@`9859727`

5.4 Use individually typed setter methods instead of a combined `set*Prameter` method

Description

Combined setter methods degrade readability and code maintainability and are prone to errors, especially when one setter method is used to store different types of values.

Examples

- GlobalConfig

code/contracts/global/GlobalConfig.sol:L18-L27

```
function setGlobalParameter(bytes32 key, uint256 value) public onlyWhitelistAdmin {
    if (key == "withdrawalLockBlockCount") {
        withdrawalLockBlockCount = value;
    } else if (key == "brokerLockBlockCount") {
        brokerLockBlockCount = value;
    } else {
        revert("key not exists");
    }
    emit UpdateGlobalParameter(key, value);
}
```

- AMMGovernance

code/contracts/liquidity/AMMGovernance.sol:L22-L42

```
function setGovernanceParameter(bytes32 key, int256 value) public onlyWhitelistAdmin
{
    if (key == "poolFeeRate") {
        governance.poolFeeRate = value.toUint256();
    } else if (key == "poolDevFeeRate") {
        governance.poolDevFeeRate = value.toUint256();
    } else if (key == "emaAlpha") {
        require(value > 0, "alpha should be > 0");
        governance.emaAlpha = value;
        emaAlpha2 = 10**18 - governance.emaAlpha;
        emaAlpha2Ln = emaAlpha2.wln();
    } else if (key == "updatePremiumPrize") {
        governance.updatePremiumPrize = value.toUint256();
    } else if (key == "markPremiumLimit") {
        governance.markPremiumLimit = value;
    } else if (key == "fundingDampener") {
        governance.fundingDampener = value;
    } else {
        revert("key not exists");
    }
    emit UpdateGovernanceParameter(key, value);
}
```

Recommendation

Implement individual setter methods for different values, especially when setting different value types.

With the current architecture multiple calls to `set*Parameter` are needed to initialize the contract. Consider adding a `constructor` or method that allows to initially set all the value with one call to save gas.

5.5 `LibTypes.Status.SETTLING` should be renamed to `LibTypes.Status.EMERGENCY`

Description

Consider renaming `LibTypes.Status.SETTLING` to `LibTypes.Status.EMERGENCY` to accurately reflect what it is being used for. The status names currently do not match the status mentioned in the specification.

code/contracts/reader/ContractReader.sol:L59-L60

```
params.isEmergency = perpetual.status() == LibTypes.Status.SETTLING;
params.isGlobalSettled = perpetual.status() == LibTypes.Status.SETTLED;
```

5.6 Implement clear, consistent naming conventions for all contracts

Description

The Mai V2 contracts do not adhere to consistent naming conventions. Because of the intricate mathematical operations and accounting inherent to the protocol, this has resulted in a significant increase in code complexity.

Addressing the underlying problem will require careful thought and consideration. Generally, the goal should be to make the contracts as readable as possible. The following list of recommendations should serve as a basis for more a more clear, consistent naming scheme within the Mai V2 contracts:

Recommendation

- Signed and unsigned variables should be distinguished: `uint uVarName` vs `int iVarName`
- Wad-denominated values should be distinguished: `uint wadVarName` vs `uint rawVarName`
- Signed and unsigned math libraries should have different names for the operations they support: `uVarName.add(...)` vs `iVarName.iAdd(...)`
- `internal` and `private` functions should be distinguished: `function _helperMethod() internal;`
- Functions that change state should never be prefixed with “get”.
    - For example: `AMM.getBuyPrice` and `AMM.getSellPrice`
- Many functions act as a wrapper for calls to either `AMM.funding` or `Perpetual.markPrice` (which calls `AMM.funding` eventually). This makes it very difficult to determine where state changes occur in the

contracts. Instead of using wrapper functions, ensure that each call to `funding` is explicit.

- For example, avoid using methods like `AMM.currentFairPrice`, which calls `funding()` (a state changing function) then `lastFairPrice()` (a `view` getter).

5.7 Prefix variables that are expected to denominated in “wads” to make them distinguishable from integers

Description

The contract system mixes raw values with values denominated in wads. Reading the code, it is not always immediately clear if a method requires or processes a wad value, or a raw value.

It is therefore recommended to prefix/suffix variables with their respective or expected type to increase code readability and maintainability and reduce the risk of variables being used in the wrong numerical context.

As one example, it is not immediately clear from calling the method `wpow` that `x` is a wad value, and `n` is a raw value. By renaming `x` to `x_wad`, its context would be much more visible.

code/contracts/lib/LibMath.sol:L103-L116

```
// x ^ n
// NOTE: n is a normal integer, do not shift 18 decimals
// solium-disable-next-line security/no-assign-params
```

```
function wpowi(int256 x, int256 n) internal pure returns (int256 z) {
    z = n % 2 != 0 ? x : _WAD;

    for (n /= 2; n != 0; n /= 2) {
        x = wmul(x, x);

        if (n % 2 != 0) {
            z = wmul(z, x);
        }
    }
}
```

5.8 Introduce a system setup phase and provide sane parameters on deployment

Description

According to the specification, the contract system can be in one of three states:

- `Normal` (**default**)
- `Emergency`
- `GlobalSettled`.

By default, after deployment, the system is in state `Normal` indicating normal operation even though the contract may not yet be fully set up for use as none of the governance settings are initialized. Uninitialized settings can lead to the system being operated in an unspecified setting and may cause all sorts of issues and side-effects.

For example, `PerpetualGovernance` is part of `Perpetual`. It allows a whitelisted admin to set critical system parameters like the `initialMarginRate` or `lotSize` which is not allowed to be zero. However, right after deployment it is uninitialized and is, therefore, going to return a zero value which is not within

specification. Since an admin is actively required to set critical parameters on a one-by-one basis it may happen that `initialiMarginRate` is never set and stays at a zero rate. This is just an example and should be easily detectable if the required processes are in place but it should be noted that there is no requirement to initialize the system with a sane configuration before it is set to `Normal` state.

code/contracts/perpetual/PerpetualGovernance.sol:L41-L44

```
governance.initialMarginRate = value.toUint256();
require(governance.initialMarginRate > 0, "require im > 0");
require(governance.initialMarginRate < 10**18, "require im < 1");
require(governance.maintenanceMarginRate < governance.initialMarginRate, "require mm
< im");
```

The general recommendation for reasonably complex systems that require parameterization before they can be set to normal operation mode is to

- provide sane (according to the specification) default values as part of the deployment process when executing the contract's `constructor`.
- introduce a one-way `Setup` phase. Make this the first phase that is active by default when deploying the contract (e.g. the first phase in the `enum`). Provide an interface for others to poll the status of the system to indicate that the system is not yet ready for normal use. In many cases it can make sense to disable certain functionality or pause the complete contract during the setup phase to reject any unwanted user interaction and minimize the risk of losses. Configure and parameterize the system as needed, perform testing to verify that it is set up according to the system deployment plan, and safely transfer it to `Normal` state indicating that it is now safe for use by others.
- do not allow to configure critical system parameters while the contract is actively being used as this can introduce unforeseeable side-effect.

5.9 Import 3rd party libraries from their original source and keep them unchanged instead of copying their content into a new library

Description

`LibMath.sol` provides two libraries `LibMathSigned` and `LibMathUnsigned`. The source unit does not contain any hints or references to the original source from where the code was taken from.

Make sure to use only security audited versions of third-party libraries with your codebase. If possible declare third-party libraries with the project's dependencies instead of copying them into your project or copying methods into new libraries. Copies of general-purpose libraries or methods may easily get outdated and often end up not being updated. This might leave the project vulnerable to security issues that are fixed in the upstream version already. Add comments for code that was taken else-where and install a process that checks 3rd party dependencies for security updates.

`LibMath.sol` contains source code from:

- https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/math/SignedSafeMath.sol
- https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/SafeCast.sol
- https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/math/Math.sol

5.10 Consider removing unnecessary events

Description

Consider removing unnecessary events like the one in GlobalConfig that just indicates that the contract has been deployed.

code/contracts/global/GlobalConfig.sol:L14-L16

```
constructor() public {
    emit CreateGlobalConfig();
}
```

5.11 Unnecessary ABIEncoderV2 declarations

Description

It should be noted that `ABIEncoderV2` is an experimental feature in Solidity `0.5.x`. With `0.6.0` the `ABIEncoderV2` is not considered experimental anymore (see solidity changelog). However, even the more recent versions of solidity list bug-fixes for the encoder and it should, therefore, be tested very thoroughly with the contract system.

To improve readability it is recommended to only specify `ABIEncoderV2` for source units that actually make use of it. For example, the following files unnecessarily declare the feature:

code/contracts/lib/LibSignature.sol:L2-L2

```
pragma experimental ABIEncoderV2; // to enable structure-type parameter
```

code/contracts/lib/LibTypes.sol:L2-L2

```
pragma experimental ABIEncoderV2; // to enable structure-type parameter
```

5.12 Avoid redefining the same structs

Description

Multiple definitions of types can be difficult to maintain and lead to security issues if the type is undergoing changes but change is not made for all definitions. Defining the same struct should, therefore, be avoided. Import the type from the respective source (e.g. `LibTypes`).

code/contracts/lib/LibSignature.sol:L4-L11

```
library LibSignature {
    enum SignatureMethod {ETH_SIGN, EIP712}

    struct OrderSignature {
        bytes32 config;
        bytes32 r;
        bytes32 s;
    }
```

code/contracts/lib/LibEIP712.sol:L3-L10

```
library LibEIP712 {
    string internal constant DOMAIN_NAME = "Mai Protocol";

    struct OrderSignature {
        bytes32 config;
        bytes32 r;
```

```
        bytes32 s;
    }
```

5.13 Methods should be declared external

Description

Review function attributes of functions that are never called from the current contract. These methods can be declared as `external` instead of `public` in order to safe gas and make the reader aware, that the method is only called by an external entity.

For example, `public` methods in `contractReader`, `PerpetualProxy`, `GlobalConfig`, `Exchange`, Governance functionality in `Perpetual`, `AMM` and other exposed API in the contract system may be declared `external`.

5.14 Gas Optimization static hashed values

Description

Pre-compute static hashed values that are known at compile-time to save some gas and add a comment describing the hashed value.

code/contracts/lib/LibEIP712.sol:L15-L16

```
bytes32 private constant EIP712_DOMAIN_TYPEHASH =
keccak256(abi.encodePacked("EIP712Domain(string name)"));
```

6 Issues

Each issue has an assigned severity:

- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Major issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

6.1 Exchange - CancelOrder has no effect Critical

Description

The exchange provides means for the `trader` or `broker` to cancel the order. The `cancelOrder` method, however, only stores the hash of the canceled order in mapping but the mapping is never checked. It is therefore effectively impossible for a trader to cancel an order.

Examples

code/contracts/exchange/Exchange.sol:L179-L187

```
function cancelOrder(LibOrder.Order memory order) public {
    require(msg.sender == order.trader || msg.sender == order.broker, "invalid
caller");

    bytes32 orderHash = order.getOrderHash();
    cancelled[orderHash] = true;

    emit Cancel(orderHash);
}
```

Recommendation

- `matchOrders*` or `validateOrderParam` should check if `cancelled[orderHash] == true` and abort fulfilling the order.
- Verify the order params (Signature) before accepting it as canceled.

6.2 AMM - `funding` can be called in emergency mode Major

Description

The specification for `AMM.funding()` states `isEmergency==FALSE` as a requirement. However, the state `isEmergency` does not exist (we assume `EMERGENCY` aka. `SETTLING`) and the implementation does not

perform any state checks. This method is called by many other functions in `AMM`.

Recommendation

According to the specification, `forceFunding` should not be allowed in `EMERGENCY` mode. However, it is assumed that this method should only be callable in `NORMAL` mode.

The assessment team would like to note that the specification appears to be inconsistent and dated (method names, variable names, …).

6.3 Perpetual - `withdraw` should only be available in `NORMAL` state Major

Description

According to the specification `withdraw` can only be called in `NORMAL` state. However, the implementation allows it to be called in `NORMAL` and `SETTLED` mode.

Examples

Withdraw only checks for `!SETTLING` state which resolves to `NORMAL` and `SETTLED`.

code/contracts/perpetual/Perpetual.sol:L175-L178

```
function withdraw(uint256 amount) public {
    withdrawFromAccount(msg.sender, amount);
}
```

code/contracts/perpetual/Perpetual.sol:L156-L169

```
function withdrawFromAccount(address payable guy, uint256 amount) private {
    require(guy != address(0), "invalid guy");
    require(status != LibTypes.Status.SETTLING, "wrong perpetual status");

    uint256 currentMarkPrice = markPrice();
    require(isSafeWithPrice(guy, currentMarkPrice), "unsafe before withdraw");
    remargin(guy, currentMarkPrice);
    address broker = currentBroker(guy);
    bool forced = broker == address(amm.perpetualProxy()) || broker == address(0);
    withdraw(guy, amount, forced);

    require(isSafeWithPrice(guy, currentMarkPrice), "unsafe after withdraw");
    require(availableMarginWithPrice(guy, currentMarkPrice) >= 0, "withdraw margin");
}
```

In contrast, `withdrawFor` requires the state to be `NORMAL`:

code/contracts/perpetual/Perpetual.sol:L171-L174

```
function withdrawFor(address payable guy, uint256 amount) public onlyWhitelisted {
    require(status == LibTypes.Status.NORMAL, "wrong perpetual status");
    withdrawFromAccount(guy, amount);
}
```

Recommendation

`withdraw` should only be available in the `NORMAL` operation mode.

6.4 Perpetual - withdrawFromInsuranceFund should check `wadAmount` instead of `rawAmount` Major

Description

`withdrawFromInsurance` checks that enough funds are in the insurance fund before allowing withdrawal by an admin by checking the provided `rawAmount <= insuranceFundBalance.toUint256()`. `rawAmount` is the `ETH` (18 digit precision) or collateral token amount (can be less than 18 digit precision) to be withdrawn while `insuranceFundBalance` is a WAD-denominated value (18 digit precision).

The check does not hold if the configured collateral has different precision and may have unwanted consequences, e.g. the withdrawal of more funds than expected.

Note: there is another check for `insuranceFundBalance` staying positive after the potential external call to collateral.

Examples

code/contracts/perpetual/Perpetual.sol:L204-L216

```
function withdrawFromInsuranceFund(uint256 rawAmount) public onlyWhitelistAdmin {
    require(rawAmount > 0, "invalid amount");
    require(insuranceFundBalance > 0, "insufficient funds");
    require(rawAmount <= insuranceFundBalance.toUint256(), "insufficient funds");

    int256 wadAmount = toWad(rawAmount);
    insuranceFundBalance = insuranceFundBalance.sub(wadAmount);
    withdrawFromProtocol(msg.sender, rawAmount);

    require(insuranceFundBalance >= 0, "negtive insurance fund");

    emit UpdateInsuranceFund(insuranceFundBalance);
}
```

When looking at the test-cases there seems to be a misconception about what unit of amount `withdrawFromInsuranceFund` is taking. For example, the insurance fund withdrawal and deposit are not tested for collateral that specifies a precision that is not 18. The test-cases falsely assume that the input to `withdrawFromInsuranceFund` is a WAD value, while it is taking the collateral's `rawAmount` which is then converted to a WAD number.

code/test/test_perpetual.js:L471-L473

```
await perpetual.withdrawFromInsuranceFund(toWad(10.111));
fund = await perpetual.insuranceFundBalance();
assert.equal(fund.toString(), 0);
```

Recommendation

Check that `require(wadAmount <= insuranceFundBalance.toUint256(), "insufficient funds");`, add a test-suite testing the insurance fund with collaterals with different precision and update existing tests that

properly provide the expected input to `withdraFromInsurance`.

6.5 Perpetual - `liquidateFrom` should not have `public` visibility Major

Description

`Perpetual.liquidate` is used to liquidate an account that is "unsafe," determined by the relative sizes of `marginBalanceWithPrice` and `maintenanceMarginWithPrice`:

code/contracts/perpetual/Perpetual.sol:L248-L253

```
// safe for liquidation
function isSafeWithPrice(address guy, uint256 currentMarkPrice) public returns (bool)
{
    return
        marginBalanceWithPrice(guy, currentMarkPrice) >=
        maintenanceMarginWithPrice(guy, currentMarkPrice).toInt256();
}
```

`Perpetual.liquidate` allows the caller to assume the liquidated account's position, as well as a small amount of "penalty collateral." The steps to liquidate are, roughly:

1. Close the liquidated account's position
2. Perform a trade on the liquidated assets with the liquidator acting as counter-party
3. Grant the liquidator a portion of the liquidated assets as a reward. An additional portion is added to the insurance fund.
4. Handle any losses

We found several issues in `Perpetual.liquidate`:

Examples

`liquidateFrom` has `public` visibility:

code/contracts/perpetual/Perpetual.sol:L270

```
function liquidateFrom(address from, address guy, uint256 maxAmount) public returns
(uint256, uint256) {
```

Given that `liquidate` only calls `liquidateFrom` after checking the current contract's status, this oversight allows anyone to call `liquidateFrom` during the `SETTLED` stage:

code/contracts/perpetual/Perpetual.sol:L291-L294

```
function liquidate(address guy, uint256 maxAmount) public returns (uint256, uint256)
{
    require(status != LibTypes.Status.SETTLED, "wrong perpetual status");
```

```
        return liquidateFrom(msg.sender, guy, maxAmount);
    }
```

Additionally, directly calling `liquidateFrom` allows anyone to liquidate on behalf of other users, forcing other accounts to assume liquidated positions.

Finally, neither `liquidate` nor `liquidateFrom` check that the liquidated account and liquidator are the same. Though the liquidation accounting process is hard to follow, we believe this is unintended and could lead to large errors in internal contract accounting.

Recommendation

- Make `liquidateFrom` an `internal` function
- In `liquidate` or `liquidateFrom`, check that `msg.sender != guy`

6.6 Unpredictable behavior due to front running or general bad timing Major

Description

In a number of cases, administrators of contracts can update or upgrade things in the system without warning. This has the potential to violate a security goal of the system.

Specifically, privileged roles could use front running to make malicious changes just ahead of incoming transactions, or purely accidental negative effects could occur due to unfortunate timing of changes.

Some instances of this are more important than others, but in general users of the system should have assurances about the behavior of the action they're about to take.

Examples

The deployer of the `PerpetualGovernance`, `AMMGovernance`, and `GlobalConfig` contracts are set as administrators for the contracts through `WhitelistedRole`. The `WhitelistedAdminRole` can whitelist other accounts at any time and allow them to perform actions protected by the `onlyWhitelisted` decorator.

Updating governance and global configuration parameters are not protected by a time-lock and take effect immediately. This, therefore, creates an opportunity for administrators to front-run users on the exchange by changing parameters for orders. It may also allow an administrator to temporarily lift restrictions for themselves (e.g. `withdrawalLockBlockCount`).

- `GlobalConfig`
  - `withdrawalLockBlockCount` is queried when applying for withdrawal. This value can be set zero enabling allowing immediate withdrawal.
  - `brokerLockBlockCount` is queried when setting a new broker. This value can e set to zero effectively enabling immediate broker changes.

code/contracts/global/GlobalConfig.sol:L18-L27

```
function setGlobalParameter(bytes32 key, uint256 value) public onlyWhitelistAdmin {
    if (key == "withdrawalLockBlockCount") {
        withdrawalLockBlockCount = value;
    } else if (key == "brokerLockBlockCount") {
        brokerLockBlockCount = value;
    } else {
        revert("key not exists");
```

```
    }
    emit UpdateGlobalParameter(key, value);
}
```

- `PerpetualGovernance`
  - e.g. Admin can front-run specific `matchOrder` calls and set arbitrary dev fees or curve parameters…

code/contracts/perpetual/PerpetualGovernance.sol:L39-L80

```
function setGovernanceParameter(bytes32 key, int256 value) public onlyWhitelistAdmin
{
    if (key == "initialMarginRate") {
        governance.initialMarginRate = value.toUint256();
        require(governance.initialMarginRate > 0, "require im > 0");
        require(governance.initialMarginRate < 10**18, "require im < 1");
        require(governance.maintenanceMarginRate < governance.initialMarginRate,
"require mm < im");
    } else if (key == "maintenanceMarginRate") {
        governance.maintenanceMarginRate = value.toUint256();
        require(governance.maintenanceMarginRate > 0, "require mm > 0");
        require(governance.maintenanceMarginRate < governance.initialMarginRate,
"require mm < im");
        require(governance.liquidationPenaltyRate < governance.maintenanceMarginRate,
"require lpr < mm");
        require(governance.penaltyFundRate < governance.maintenanceMarginRate,
"require pfr < mm");
    } else if (key == "liquidationPenaltyRate") {
        governance.liquidationPenaltyRate = value.toUint256();
        require(governance.liquidationPenaltyRate < governance.maintenanceMarginRate,
"require lpr < mm");
    } else if (key == "penaltyFundRate") {
        governance.penaltyFundRate = value.toUint256();
        require(governance.penaltyFundRate < governance.maintenanceMarginRate,
"require pfr < mm");
    } else if (key == "takerDevFeeRate") {
        governance.takerDevFeeRate = value;
    } else if (key == "makerDevFeeRate") {
        governance.makerDevFeeRate = value;
    } else if (key == "lotSize") {
        require(
            governance.tradingLotSize == 0 ||
governance.tradingLotSize.mod(value.toUint256()) == 0,
            "require tls % ls == 0"
        );
        governance.lotSize = value.toUint256();
    } else if (key == "tradingLotSize") {
        require(governance.lotSize == 0 || value.toUint256().mod(governance.lotSize)
== 0, "require tls % ls == 0");
        governance.tradingLotSize = value.toUint256();
    } else if (key == "longSocialLossPerContracts") {
        require(status == LibTypes.Status.SETTLING, "wrong perpetual status");
```

```
        socialLossPerContracts[uint256(LibTypes.Side.LONG)] = value;
    } else if (key == "shortSocialLossPerContracts") {
        require(status == LibTypes.Status.SETTLING, "wrong perpetual status");
        socialLossPerContracts[uint256(LibTypes.Side.SHORT)] = value;
    } else {
        revert("key not exists");
    }
    emit UpdateGovernanceParameter(key, value);
}
```

- Admin can set `devAddress` or even update to a new `amm` and `globalConfig`

code/contracts/perpetual/PerpetualGovernance.sol:L82-L94

```
function setGovernanceAddress(bytes32 key, address value) public onlyWhitelistAdmin {
    require(value != address(0x0), "invalid address");
    if (key == "dev") {
        devAddress = value;
    } else if (key == "amm") {
        amm = IAMM(value);
    } else if (key == "globalConfig") {
        globalConfig = IGlobalConfig(value);
    } else {
        revert("key not exists");
    }
    emit UpdateGovernanceAddress(key, value);
}
```

- `AMMGovernance`

code/contracts/liquidity/AMMGovernance.sol:L22-L43

```
function setGovernanceParameter(bytes32 key, int256 value) public onlyWhitelistAdmin
{
    if (key == "poolFeeRate") {
        governance.poolFeeRate = value.toUint256();
    } else if (key == "poolDevFeeRate") {
        governance.poolDevFeeRate = value.toUint256();
    } else if (key == "emaAlpha") {
        require(value > 0, "alpha should be > 0");
        governance.emaAlpha = value;
        emaAlpha2 = 10**18 - governance.emaAlpha;
        emaAlpha2Ln = emaAlpha2.wln();
    } else if (key == "updatePremiumPrize") {
        governance.updatePremiumPrize = value.toUint256();
    } else if (key == "markPremiumLimit") {
        governance.markPremiumLimit = value;
    } else if (key == "fundingDampener") {
        governance.fundingDampener = value;
    } else {
```

```
            revert("key not exists");
        }
        emit UpdateGovernanceParameter(key, value);
    }
```

Recommendation

The underlying issue is that users of the system can’t be sure what the behavior of a function call will be, and this is because the behavior can change at any time.

We recommend giving the user advance notice of changes with a time lock. For example, make all updates to system parameters or upgrades require two steps with a mandatory time window between them. The first step merely broadcasts to users that a particular change is coming, and the second step commits that change after a suitable waiting period.

Additionally, users should verify the whitelist setup before using the contract system and monitor it for new additions to the whitelist. Documentation should clearly outline what roles are owned by whom to support suitability. Sane parameter bounds should be enforced (e.g. min. disallow block delay of zero)

6.7 AMM - Governance is able to set an invalid alpha value Medium

Description

According to https://en.wikipedia.org/wiki/Moving_average

> *The coefficient α represents the degree of weighting decrease, a constant smoothing factor between 0 and 1. A higher α discounts older observations faster.*

However, the code does not check upper bounds. An admin may, therefore, set an invalid alpha that puts `emaAlpha2` out of bounds or negative.

Examples

code/contracts/liquidity/AMMGovernance.sol:L27-L31

```
} else if (key == "emaAlpha") {
    require(value > 0, "alpha should be > 0");
    governance.emaAlpha = value;
    emaAlpha2 = 10**18 - governance.emaAlpha;
    emaAlpha2Ln = emaAlpha2.wln();
```

Recommendation

Ensure that the system configuration is always within safe bounds. Document expected system variable types and their safe operating ranges. Enforce that bounds are checked every time a value is set. Enforce safe defaults when deploying contracts.

Ensure `emaAlpha` is `0 < value < 1 WAD`

6.8 AMM - Amount of collateral spent or shares received may be unpredictable for liquidity provider Medium

Description

When providing liquidity with `addLiquidity()`, the amount of collateral required is based on the current price and the amount of shares received depends on the total amount of shares in circulation. This price can fluctuate at a moment's notice, making the behavior of the function unpredictable for the user.

The same is true when removing liquidity via `removeLiquidity()`.

Recommendation

Unpredictability can be introduced by someone front-running the transaction, or simply by poor timing. For example, adjustments to global variable configuration by the system admin will directly impact subsequent actions by the user. In order to ensure users know what to expect:

- Allow the caller to specify a price limit or maximum amount of collateral to be spent
- Allow the caller to specify the minimum amount of shares expected to be received

6.9 Exchange - insufficient input validation in `matchOrders` Medium

Description

`matchOrders` does not check that that the sender has provided the same number of `amounts` as `makerOrderParams`. When fewer `amounts` exist than `makerOrderParams`, the method will revert because of an out-of-bounds array access. When fewer `makerOrderParams` exist than `amounts`, the method will succeed, and the additional values in `amounts` will be ignored.

Additionally, the method allows the sender to provide no `makerOrderParams` at all, resulting in no state changes.

`matchOrders` also does not reject trades with an amount set to zero. Such orders should be rejected because they do not comply with the minimum `tradingLotSize` configured for the system. As a side-effect, events may be emitted for zero-amount trades and unexpected state changes may occur.

Examples

code/contracts/exchange/Exchange.sol:L34-L39

```
function matchOrders(
    LibOrder.OrderParam memory takerOrderParam,
    LibOrder.OrderParam[] memory makerOrderParams,
    address _perpetual,
    uint256[] memory amounts
) public {
```

code/contracts/exchange/Exchange.sol:L113-L113

```
function matchOrderWithAMM(LibOrder.OrderParam memory takerOrderParam, address
_perpetual, uint256 amount) public {
```

Recommendation

- Require `makerOrderParams.length > 0 && amounts.length == makerOrderParams.length`
- Require that `amount` or any of the `amounts[i]` provided to `matchOrders` is `>=tradingLotSize`.

6.10 AMM - Liquidity provider may lose up to `lotSize` when removing liquidity Medium

Description

When removing liquidity, the amount of collateral received is calculated from the `shareAmount` (ShareToken) of the liquidity provider. The liquidity removal process registers a trade on the amount, with the liquidity provider and `AMM` taking opposite sides. Because trading only accepts multiple of the `lotSize`, the leftover is discarded. The amount discarded may be up to `lotSize - 1`.

The expectation is that this value should not be too high, but as `lotSize` can be set to arbitrary values by an admin, it is possible that this step discards significant value. Additionally, see https://github.com/Consensys/mcdexio-mai-protocol-v2-audit-2020-05/issues/16 for how this can be exploited by an admin.

Note that similar behavior is present in `Perpetual.liquidateFrom`, where the `liquidatableAmount` calculated undergoes a similar modulo operation:

code/contracts/perpetual/Perpetual.sol:L277-L278

```
uint256 liquidatableAmount =
totalPositionSize.sub(totalPositionSize.mod(governance.lotSize));
liquidationAmount =
liquidationAmount.ceil(governance.lotSize).min(maxAmount).min(liquidatableAmount);
```

Examples

- `lotSize` can arbitrarily be set up to `pos_int256_max` as long as `tradingLotSize % lotSize == 0`

code/contracts/perpetual/PerpetualGovernance.sol:L61-L69

```
} else if (key == "lotSize") {
    require(
        governance.tradingLotSize == 0 ||
governance.tradingLotSize.mod(value.toUint256()) == 0,
        "require tls % ls == 0"
    );
    governance.lotSize = value.toUint256();
} else if (key == "tradingLotSize") {
    require(governance.lotSize == 0 || value.toUint256().mod(governance.lotSize) ==
0, "require tls % ls == 0");
    governance.tradingLotSize = value.toUint256();
```

- `amount` is derived from `shareAmount` rounded down to the next multiple of the `lotSize`. The leftover is discarded.

code/contracts/liquidity/AMM.sol:L289-L294

```
uint256 amount =
shareAmount.wmul(oldPoolPositionSize).wdiv(shareToken.totalSupply());
amount = amount.sub(amount.mod(perpetualProxy.lotSize()));

perpetualProxy.transferBalanceOut(trader, price.wmul(amount).mul(2));
burnShareTokenFrom(trader, shareAmount);
uint256 opened = perpetualProxy.trade(trader, LibTypes.Side.LONG, price, amount);
```

Recommendation

- Ensure that documentation makes users aware of the fact that they may lose up to `lotsize-1` in value.

- Alternatively, track accrued value and permit trades on values that exceed `lotSize`. Note that this may add significant complexity.
- Ensure that similar system behavior, like the `liquidatableAmount` calculated in `Perpetual.liquidateFrom`, is also documented and communicated clearly to users.

6.11 Oracle - Unchecked oracle response timestamp and integer over/underflow Medium

Description

The external Chainlink oracle, which provides index price information to the system, introduces risk inherent to any dependency on third-party data sources. For example, the oracle could fall behind or otherwise fail to be maintained, resulting in outdated data being fed to the index price calculations of the AMM. Oracle reliance has historically resulted in crippled on-chain systems, and complications that lead to these outcomes can arise from things as simple as network congestion.

Ensuring that unexpected oracle return values are properly handled will reduce reliance on off-chain components and increase the resiliency of the smart contract system that depends on them.

Examples

1. The `ChainlinkAdapter` and `InversedChainlinkAdapter` take the oracle's (int256) `latestAnswer` and convert the result using `chainlinkDecimalsAdapter`. This arithmetic operation can underflow/overflow if the Oracle provides a large enough answer:

code/contracts/oracle/ChainlinkAdapter.sol:L10-L19

```
int256 public constant chainlinkDecimalsAdapter = 10**10;

constructor(address _feeder) public {
    feeder = IChainlinkFeeder(_feeder);
}

function price() public view returns (uint256 newPrice, uint256 timestamp) {
    newPrice = (feeder.latestAnswer() * chainlinkDecimalsAdapter).toUint256();
    timestamp = feeder.latestTimestamp();
}
```

code/contracts/oracle/InversedChainlinkAdapter.sol:L11-L20

```
int256 public constant chainlinkDecimalsAdapter = 10**10;

constructor(address _feeder) public {
    feeder = IChainlinkFeeder(_feeder);
}

function price() public view returns (uint256 newPrice, uint256 timestamp) {
    newPrice = ONE.wdiv(feeder.latestAnswer() *
chainlinkDecimalsAdapter).toUint256();
    timestamp = feeder.latestTimestamp();
}
```

1. The oracle provides a timestamp for the `latestAnswer` that is not validated and may lead to old oracle timestamps being accepted (e.g. caused by congestion on the blockchain or a directed censorship attack).

code/contracts/oracle/InversedChainlinkAdapter.sol:L19-L20

```
    timestamp = feeder.latestTimestamp();
}
```

Recommendation

- Use `SafeMath` for mathematical computations
- Verify `latestAnswer` is within valid bounds (`!=0`)
- Verify `latestTimestamp` is within accepted bounds (not in the future, was updated within a reasonable amount of time)
- Deduplicate code by combining both Adapters into one as the only difference is that the `InversedChainlinkAdapter` returns `ONE.wdiv(price)` .

6.12 AMM - Liquidity pools can be initialized with zero collateral Medium

Description

`createPool` can be initialized with `amount == 0`. Because a subsequent call to `initFunding` can only happen once, the contract is now initialized with a zero size pool that does not allow any liquidity to be added.

Trying to recover by calling `createPool` again fails as the funding state is already `initialized`. The specification also states the following about `createPool`:

> *Open asset pool by deposit to AMM. Only available when pool is empty.*

This is inaccurate, as `createPool` can only be called once due to a check in `initFunding`, but this call may leave the pool empty.

Furthermore, the contract's liquidity management functionality (`addLiquidity` and `removeLiquidity`) allows adding zero liquidity (`amount == 0`) and removing zero shares (`shareAmount == 0`). As these actions do not change the liquidity of the pool, they should be rejected.

Recommendation

- Require a minimum amount `lotSize` to be provided when creating a Pool and adding liquidity via `addLiquidity`
- Require a minimum amount of shares to be provided when removing liquidity via `removeLiquidity`

6.13 Perpetual - Administrators can put the system into emergency mode indefinitely Medium

Description

There is no limitation on how long an administrator can put the `Perpetual` contract into emergency mode. Users cannot trade or withdraw funds in emergency mode and are effectively locked out until the admin chooses

to put the contract in `SETTLED` mode.

Examples

code/contracts/perpetual/PerpetualGovernance.sol:L96-L101

```
function beginGlobalSettlement(uint256 price) public onlyWhitelistAdmin {
    require(status != LibTypes.Status.SETTLED, "already settled");
    settlementPrice = price;
    status = LibTypes.Status.SETTLING;
    emit BeginGlobalSettlement(price);
}
```

code/contracts/perpetual/Perpetual.sol:L146-L154

```
function endGlobalSettlement() public onlyWhitelistAdmin {
    require(status == LibTypes.Status.SETTLING, "wrong perpetual status");

    address guy = address(amm.perpetualProxy());
    settleFor(guy);
    status = LibTypes.Status.SETTLED;

    emit EndGlobalSettlement();
}
```

Recommendation

- Set a time-lock when entering emergency mode that allows anyone to set the system to `SETTLED` after a fixed amount of time.

6.14 Signed data may be usable cross-chain Medium

Description

Signed order data may be re-usable cross-chain as the chain-id is not explicitly part of the signed data.

It is also recommended to further harden the signature verification and validate that `v` and `s` are within expected bounds. `ecrecover()` returns `0x0` to indicate an error condition, therefore, a `signerAddress` or `recovered` address of `0x0` should explicitly be disallowed.

Examples

The signed order data currently includes the EIP712 Domain Name `Mai Protocol` and the following information:

code/contracts/lib/LibOrder.sol:L23-L48

```
struct Order {
    address trader;
    address broker;
    address perpetual;
    uint256 amount;
    uint256 price;
    /**
     * Data contains the following values packed into 32 bytes
     *
╔═══════════════════╤═══════════════════════════════════════════════════════════════════════
     * ║                   | length(bytes)   desc
                                                                                              ║
     *
╟───────────────────┼───────────────────────────────────────────────────────────────────────
     * ║ version           | 1               order version
                                                                                              ║
     * ║ side              | 1               0: buy (long), 1: sell (short)
                                                                                              ║
     * ║ isMarketOrder     | 1               0: limitOrder, 1: marketOrder
                                                                                              ║
     * ║ expiredAt         | 5               order expiration time in seconds
                                                                                              ║
     * ║ asMakerFeeRate    | 2               maker fee rate (base 100,000)
                                                                                              ║
     * ║ asTakerFeeRate    | 2               taker fee rate (base 100,000)
                                                                                              ║
     * ║ (d) makerRebateRate| 2              rebate rate for maker (base 100)
                                                                                              ║
     * ║ salt              | 8               salt
                                                                                              ║
     * ║ isMakerOnly       | 1               is maker only
                                                                                              ║
     * ║ isInversed        | 1               is inversed contract
                                                                                              ║
     * ║                   | 8               reserved
                                                                                              ║
     *
╚═══════════════════╧═══════════════════════════════════════════════════════════════════════

     */
    bytes32 data;
}
```

Signature verification:

code/contracts/lib/LibSignature.sol:L24-L47

```
function isValidSignature(OrderSignature memory signature, bytes32 hash, address
signerAddress)
    internal
    pure
    returns (bool)
{
    uint8 method = uint8(signature.config[1]);
    address recovered;
    uint8 v = uint8(signature.config[0]);

    if (method == uint8(SignatureMethod.ETH_SIGN)) {
        recovered = ecrecover(
            keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n32", hash)),
            v,
            signature.r,
            signature.s
        );
    } else if (method == uint8(SignatureMethod.EIP712)) {
        recovered = ecrecover(hash, v, signature.r, signature.s);
    } else {
        revert("invalid sign method");
    }

    return signerAddress == recovered;
}
```

Recommendation

- Include the `chain-id` in the signature to avoid cross-chain validity of signatures
- verify `s` is within valid bounds to avoid signature malleability

```
if (uint256(s) >
0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0) {
  revert("ECDSA: invalid signature 's' value");
}
```

- verify `v` is within valid bounds

```
if (v != 27 && v != 28) {
 revert("ECDSA: invalid signature 'v' value");
}
```

- return invalid if the result of `ecrecover()` is `0x0`

6.15 Exchange - validateOrderParam does not check against `SUPPORTED_ORDER_VERSION` Medium

Description

`validateOrderParam` verifies the signature and version of a provided order. Instead of checking against the contract constant `SUPPORTED_ORDER_VERSION` it, however, checks against a hardcoded version `2` in the method itself.

This might be a problem if `SUPPORTED_ORDER_VERSION` is seen as the configuration parameter for the allowed version. Changing it would not change the allowed order version for `validateOrderParam` as this constant literal is never used.

At the time of this audit, however, the `SUPPORTED_ORDER_VERSION` value equals the hardcoded value in the `validateOrderParam` method.

Examples

code/contracts/exchange/Exchange.sol:L155-L170

```
function validateOrderParam(IPerpetual perpetual, LibOrder.OrderParam memory
orderParam)
    internal
    view
    returns (bytes32)
{
    address broker = perpetual.currentBroker(orderParam.trader);
    require(broker == msg.sender, "invalid broker");
    require(orderParam.getOrderVersion() == 2, "unsupported version");
    require(orderParam.getExpiredAt() >= block.timestamp, "order expired");

    bytes32 orderHash = orderParam.getOrderHash(address(perpetual), broker);
    require(orderParam.signature.isValidSignature(orderHash, orderParam.trader),
"invalid signature");
    require(filled[orderHash] < orderParam.amount, "fullfilled order");

    return orderHash;
}
```

Recommendation

Check against `SUPPORTED_ORDER_VERSION` instead of the hardcoded value `2`.

6.16 LibMathSigned - `wpowi` returns an invalid result for a negative exponent Medium

Description

`LibMathSigned.wpowi(x,n)` calculates Wad value `x` (base) to the power of `n` (exponent). The exponent is declared as a signed int, however, the method returns wrong results when calculating `x ^(-n)`.

The comment for the `wpowi` method suggests that `n` is a normal integer instead of a Wad-denominated value. This, however, is not being enforced.

Examples

- `LibMathSigned.wpowi(8000000000000000000, 2) = 64000000000000000000`
- (wrong) `LibMathSigned.wpowi(8000000000000000000, -2) = 64000000000000000000`

code/contracts/lib/LibMath.sol:L103-L116

```
// x ^ n
// NOTE: n is a normal integer, do not shift 18 decimals
// solium-disable-next-line security/no-assign-params
function wpowi(int256 x, int256 n) internal pure returns (int256 z) {
    z = n % 2 != 0 ? x : _WAD;

    for (n /= 2; n != 0; n /= 2) {
        x = wmul(x, x);

        if (n % 2 != 0) {
            z = wmul(z, x);
        }
    }
}
```

Recommendation

Make `wpowi` support negative exponents or use the proper type for `n` (`uint`) and reject negative values.

Enforce that the exponent bounds are within sane ranges and less than a Wad to detect potential misuse where someone accidentally provides a Wad value as `n` .

Add positive and negative unit-tests to fully cover this functionality.

6.17 Outdated solidity version and floating pragma Medium

Description

Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues (see also https://github.com/ethereum/solidity/releases) that affect the current compiler version.

The codebase specifies a floating version of `^0.5.2` and makes use of the experimental feature `ABIEncoderV2` .

It should be noted, that `ABIEncoderV2` was subject to multiple bug-fixes up until the latest `0.6.x` version and contracts compiled with earlier versions are - for example - susceptible to the following issues:

- ImplicitConstructorCallvalueCheck
- TupleAssignmentMultiStackSlotComponents
- MemoryArrayCreationOverflow
- privateCanBeOverridden
- YulOptimizerRedundantAssignmentBreakContinue0.5
- ABIEncoderV2CalldataStructsWithStaticallySizedAndDynamicallyEncodedMembers
- SignedArrayStorageCopy
- ABIEncoderV2StorageArrayWithMultiSlotElement
- DynamicConstructorArgumentsClippedABIV2

Examples

Codebase declares compiler version `^0.5.2`:

code/contracts/liquidity/AMM.sol:L1-L2

```
pragma solidity ^0.5.2;
pragma experimental ABIEncoderV2; // to enable structure-type parameters
```

According to etherscan.io, the currently deployed main-net `AMM` contract is compiled with solidity version `0.5.8`:

https://etherscan.io/address/0xb95B9fb0539Ec84DeD2855Ed1C9C686Af9A4e8b3#code

Recommendation

It is recommended to settle on the latest stable 0.6.x or 0.5.x version of the Solidity compiler and lock the pragma version to a specifically tested compiler release.

6.18 AMM - ONE_WAD_U is never used Minor

Description

The const `ONE_WAD_U` is declared but never used. Avoid re-declaring the same constants in multiple source-units (and unit-test cases) as this will be hard to maintain.

Examples

code/contracts/liquidity/AMM.sol:L17-L17

```
uint256 private constant ONE_WAD_U = 10**18;
```

Recommendation

Remove unused code. Import the value from a shared resource. E.g. `ONE_WAD` is declared multiple times in `LibMathSigned`, `LibMathUnsigned`, `AMM`, hardcoded in checks in `PerpetualGovernance.setGovernanceParameter`, `AMMGovernance.setGovernanceParameter`.

6.19 Perpetual - Variable shadowing in constructor Minor

Description

`Perpetual` inherits from `PerpetualGovernance` and `Collateral`, which declare state variables that are shadowed in the `Perpetual` constructor.

Examples

- Local constructor argument shadows `PerpetualGovernance.globalConfig`, `PerpetualGovernance.devAddress`, `Collateral.collateral`

Note: Confusing name: `Collateral` is an inherited contract and a state variable.

code/contracts/perpetual/Perpetual.sol:L34-L41

```
constructor(address globalConfig, address devAddress, address collateral, uint256
collateralDecimals)
    public
    Position(collateral, collateralDecimals)
{
    setGovernanceAddress("globalConfig", globalConfig);
    setGovernanceAddress("dev", devAddress);
    emit CreatePerpetual();
}
```

Recommendation

Rename the parameter or state variable.

6.20 Perpetual - The specified decimals for the collateral may not reflect the token's actual decimals Minor

Description

When initializing the `Perpetual` contract, the deployer can decide to use either `ETH`, or an `ERC20`-compliant collateral. In the latter case, the deployer must provide a nonzero address for the token, as well as the number of `decimals` used by the token:

code/contracts/perpetual/Collateral.sol:L28-L34

```
constructor(address _collateral, uint256 decimals) public {
    require(decimals <= MAX_DECIMALS, "decimals out of range");
    require(_collateral != address(0x0) || (_collateral == address(0x0) && decimals
== 18), "invalid decimals");

    collateral = _collateral;
    scaler = (decimals == MAX_DECIMALS ? 1 : 10**(MAX_DECIMALS -
decimals)).toInt256();
}
```

The provided `decimals` value is not checked for validity and can differ from the actual token's decimals.

Recommendation

Ensure to establish documentation that makes users aware of the fact that the decimals configured are not enforced to match the actual tokens decimals. This is to allow users to audit the system configuration and decide whether they want to participate in it.

6.21 AMM - Unchecked return value in `ShareToken.mint` Minor

Description

`ShareToken` is an extension of the Openzeppelin ERC20Mintable pattern which exposes a method called `mint()` that allows accounts owning the minter role to mint new tokens. The return value of `ShareToken.mint()` is not checked.

Since the ERC20 standard does not define whether this method should return a value or revert it may be problematic to assume that all tokens revert. If, for example, an implementation is used that does not revert on error but returns a boolean error indicator instead the caller might falsely continue without the token minted.

We would like to note that the functionality is intended to be used with the provided `ShareToken` and therefore the contract is safe to use assuming `ERC20Mintable.mint` reverts on error. The issue arises if the system is used with a different `ShareToken` implementation that is not implemented in the same way.

Examples

- Openzeppelin implementation

```
function mint(address account, uint256 amount) public onlyMinter returns (bool) {
_mint(account, amount);
return true;
}
```

- Call with unchecked return value

code/contracts/liquidity/AMM.sol:L499-L502

```
function mintShareTokenTo(address guy, uint256 amount) internal {
    shareToken.mint(guy, amount);
}
```

Recommendation

Consider wrapping the `mint` statement in a `require` clause, however, this way only tokens that are returning a boolean error indicator are supported. Document the specification requirements for the `ShareToken` and clearly state if the token is expected to revert or return an error indicator.

It should also be documented that the Token exposes a `burn` method that does not adhere to the Openzeppelin `ERC20Burnable` implementation. The `ERC20Burnable` import is unused as noted in https://github.com/Consensys/mcdexio-mai-protocol-v2-audit-2020-05/issues/18.

6.22 Perpetual - `beginGlobalSettlement` can be called multiple times Minor

Description

The system can be put into emergency mode by an admin calling `beginGlobalSettlement` and providing a fixed `settlementPrice`. The method can be invoked even when the contract is already in `SETTLING` (emergency) mode, allowing an admin to selectively adjust the settlement price again. This does not seem to be the intended behavior as calling the method again re-sets the status to `SETTLING`. Furthermore, it may affect users' behavior during the SETTLING phase.

Examples

code/contracts/perpetual/PerpetualGovernance.sol:L96-L101

```
function beginGlobalSettlement(uint256 price) public onlyWhitelistAdmin {
    require(status != LibTypes.Status.SETTLED, "already settled");
    settlementPrice = price;
    status = LibTypes.Status.SETTLING;
    emit BeginGlobalSettlement(price);
}
```

Recommendation

- Emergency mode should only be allowed to be set once

6.23 Unused Imports Minor

Description

The following source units are imported but not referenced in the contract:

Examples

code/contracts/perpetual/Perpetual.sol:L4-L5

```
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
```

code/contracts/perpetual/Perpetual.sol:L14-L15

```
import "../interface/IPriceFeeder.sol";
import "../interface/IGlobalConfig.sol";
```

code/contracts/token/ShareToken.sol:L5-L5

```
import "@openzeppelin/contracts/token/ERC20/ERC20Burnable.sol";
```

code/contracts/token/ShareToken.sol:L3-L3

```
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
```

Recommendation

Check all imports and remove all unused/unreferenced and unnecessary imports.

6.24 Exchange - OrderStatus is never used Minor

Description

The enum `OrderStatus` is declared but never used.

Examples

code/contracts/exchange/Exchange.sol:L20-L20

```
enum OrderStatus {EXPIRED, CANCELLED, FILLABLE, FULLY_FILLED}
```

Recommendation

Remove unused code.

6.25 LibMath - Inaccurate declaration of `_UINT256_MAX` Minor

Description

`LibMathUnsigned` declares `_UINT256_MAX` as `2^255-1` while this value actually represents `_INT256_MAX`. This appears to just be a naming issue.

Examples

(`UINT256_MAX/2-1 => pos INT256_MAX` ; `2**256/2-1==2**255-1`)

code/contracts/lib/LibMath.sol:L228-L230

```
library LibMathUnsigned {
    uint256 private constant _WAD = 10**18;
    uint256 private constant _UINT256_MAX = 2**255 - 1;
```

Recommendation

Rename `_UINT256_MAX` to `_INT256MAX` or `_SIGNED_INT256MAX`.

6.26 LibMath - inconsistent assertion text and improve representation of literals with many digits Minor

Description

The assertion below states that `logE only accepts v <= 1e22 * 1e18` while the argument name is `x`. In addition to that we suggest representing large literals in scientific notation.

Examples

code/contracts/lib/LibMath.sol:L153-L157

```
function wln(int256 x) internal pure returns (int256) {
    require(x > 0, "logE of negative number");
    require(x <= 100000000000000000000000000000000000000000, "logE only accepts v <=
1e22 * 1e18"); // in order to prevent using safe-math
    int256 r = 0;
    uint8 extra_digits = longer_digits - fixed_digits;
```

Recommendation

Update the inconsistent assertion text `v` -> `x` and represent large literals in scientific notation as they are otherwise difficult to read and review.

6.27 LibMath - roundHalfUp returns unfinished result Minor

Description

The method `LibMathSigned.roundHalfUp(int x, int y)` returns the value `x` rounded up to the base `y`. The method suggests that the result is the rounded value while that's not actually true. The result for a positive `x` is `x + base/2` and `x - base/2` for negative values. The rounding is not yet finished as this would require a final division by base `y` to manifest the rounding.

It is assumed that the final rounding step is not executed for performance reasons. However, this might easily introduce errors when the caller assumes the result is rounded for base while it is not.

Examples

- `roundHalfUp(-4700, 1000) = -4700` instead of `5000`
- `roundHalfUp(4700, 1000) = 4700` instead of `5000`

code/contracts/lib/LibMath.sol:L126-L133

```
// ROUND_HALF_UP rule helper. 0.5 ≈ 1, 0.4 ≈ 0, -0.5 ≈ -1, -0.4 ≈ 0
function roundHalfUp(int256 x, int256 y) internal pure returns (int256) {
    require(y > 0, "roundHalfUp only supports y > 0");
    if (x >= 0) {
        return add(x, y / 2);
    }
    return sub(x, y / 2);
}
```

Recommendation

We have verified the current code-base and the callers for `roundHalfUp` are correctly finishing the rounding step. However, it is recommended to finish the rounding within the method or document this behavior to prevent errors caused by code that falsely assumes that the returned value finished rounding.

6.28 LibMath/LibOrder - unused named return value Minor

Description

The following methods declare a named return value but explicitly return a value instead. The named return value is not used.

- `LibMathSigned.min()`
- `LibMathSigned.max()`
- `LibMathUnsigned.min()`
- `LibMathUnsigned.max()`
- `LibOrder.getOrderHash()`
- `LibOrder.hashOrder()`

Examples

code/contracts/lib/LibMath.sol:L90-L96

```
function min(int256 x, int256 y) internal pure returns (int256 z) {
    return x <= y ? x : y;
}

function max(int256 x, int256 y) internal pure returns (int256 z) {
    return x >= y ? x : y;
}
```

code/contracts/lib/LibMath.sol:L285-L292

```
function min(uint256 x, uint256 y) internal pure returns (uint256 z) {
    return x <= y ? x : y;
}

function max(uint256 x, uint256 y) internal pure returns (uint256 z) {
    return x >= y ? x : y;
}
```

code/contracts/lib/LibOrder.sol:L68-L71

```
function getOrderHash(Order memory order) internal pure returns (bytes32 orderHash) {
    orderHash = LibEIP712.hashEIP712Message(hashOrder(order));
    return orderHash;
}
```

code/contracts/lib/LibOrder.sol:L86-L97

```
function hashOrder(Order memory order) internal pure returns (bytes32 result) {
    bytes32 orderType = EIP712_ORDER_TYPE;
    // solium-disable-next-line security/no-inline-assembly
    assembly {
        let start := sub(order, 32)
        let tmp := mload(start)
        mstore(start, orderType)
        result := keccak256(start, 224)
        mstore(start, tmp)
    }
    return result;
}
```

Recommendation

Remove the named return value and explicitly return the value.

6.29 Where possible, a specific contract type should be used rather than `address` Minor

Description

Rather than storing `address`es and then casting to the known contract type, it's better to use the best type available so the compiler can check for type safety.

Examples

`Collateral. collateral` is of type `address`, but it could be type `IERC20` instead. Not only would this give a little more type safety when deploying new modules, but it would avoid repeated casts throughout the codebase of the form `IERC20(collateral)`, `IPerpetual(_perpetual)` and others. The following is an incomplete list of examples:

- declare collateral as IERC20

code/contracts/perpetual/Collateral.sol:L19-L19

```
address public collateral;
```

code/contracts/perpetual/Collateral.sol:L51-L51

```
IERC20(collateral).safeTransferFrom(guy, address(this), rawAmount);
```

- declare argument `perpetual` as `IPerpetual`

code/contracts/exchange/Exchange.sol:L34-L42

```
function matchOrders(
    LibOrder.OrderParam memory takerOrderParam,
    LibOrder.OrderParam[] memory makerOrderParams,
    address _perpetual,
    uint256[] memory amounts
```

```
) public {
    require(!takerOrderParam.isMakerOnly(), "taker order is maker only");

    IPerpetual perpetual = IPerpetual(_perpetual);
```

- declare argument `feeder` as `IChainlinkFeeder`

code/contracts/oracle/ChainlinkAdapter.sol:L12-L14

```
constructor(address _feeder) public {
    feeder = IChainlinkFeeder(_feeder);
}
```

Remediation

Where possible, use more specific types instead of `address`. This goes for parameter types as well as state variable types.

Appendix 1 - Files in Scope

This audit covered the following files of the mcdexio/mai-protocol-v2 source code repository:

File Name	SHA-1 Hash
contracts/token/ShareToken.sol	381ad1be612285ad2396bf157377721e285ed2fc
contracts/reader/ContractReader.sol	6177fd113dc02f6865bc1670e060a048c1ccccbd
contracts/perpetual/Brokerage.sol	cb5096494e9069652c8734065d24eb94fc0bff6a
contracts/perpetual/Position.sol	5591403e58a6275423e775e0ba0bd791a9e984de
contracts/perpetual/PerpetualGovernance.sol	814c9d9968fcc6c9ee277b2c6dd7ed7fceb0e579
contracts/perpetual/Collateral.sol	22a68f571f18ea64ed13abf7890ecc1142915319
contracts/perpetual/Perpetual.sol	1a21792905c2b539250eac8dff23d26f41b8857e
contracts/interface/IPerpetual.sol	1e01454b6d0ba5b87b6c04f49603ea7707493df0
contracts/interface/IPerpetualProxy.sol	d299c3517f5ebbea7084e7164132fd8dbafdd4e0
contracts/interface/IPriceFeeder.sol	b3bda1d692e4ed8695645fcae9c477912b589d55
contracts/interface/IGlobalConfig.sol	3c0938eddfa01b89441711abe96e6ae526447449
contracts/interface/IChainlinkFeeder.sol	ddcb0abf1ee1340a0f17efe759a51735325a5994
contracts/interface/IAMM.sol	1868a2ecaee45cc12bda6008a0b3d75e2bdc9630
contracts/exchange/Exchange.sol	997aac71ecce4cdb5d41e55a4dd3e96f1a2aa36a
contracts/proxy/PerpetualProxy.sol	a210d447cfb54001d1038d42b1114baf021e4cb2
contracts/oracle/InversedChainlinkAdapter.sol	0e2dbd0a887083c2908136437dea938d0d4c4df2
contracts/oracle/ChainlinkAdapter.sol	a8df0adf24497f4310fd8beab999b7750c599261
contracts/lib/LibSignature.sol	d172e939094fe391a7f1854719791d352f56e63f
contracts/lib/LibOrder.sol	606607a62692cc6067177d605b3bba0215211bd7
contracts/lib/LibEIP712.sol	6cb47eb26501eb52fbb83d1cd8d3b7e8425d5e0d
contracts/lib/LibTypes.sol	bc18a1473b87daa954aeb89fa5fb6c39aa673c40
contracts/lib/LibMath.sol	f81580d413630756486ac49f8de85320e93183d3
contracts/global/GlobalConfig.sol	8450d1c86e45a03d8b761086274af0ce00ac38b2
contracts/liquidity/AMM.sol	bc103b3b4079014b706f7e1c926e46ad60c4c824

File Name	SHA-1 Hash
contracts/liquidity/AMMGovernance.sol	88c156db493915ec2749eaf68832bfc2d21f5164

Excluded Source Units

File
contracts/test/TestOrder.sol
contracts/test/TestToken.sol
contracts/test/TestBrokerage.sol
contracts/test/TestAMM.sol
contracts/test/TestCollateral.sol
contracts/test/TestPerpetual.sol
contracts/test/TestFundingMock.sol
contracts/test/TestPosition.sol
contracts/test/TestPerpetualGovernance.sol
contracts/test/TestMath.sol
contracts/test/TestSignature.sol
contracts/test/TestPriceFeeder.sol
contracts/test/TestTypes.sol

Appendix 2 - Artifacts

This section contains some of the artifacts generated during our review by automated tools, the test suite, etc. If any issues or recommendations were identified by the output presented here, they have been addressed in the appropriate section above.

A.2.1 Solidity Code Metrics

Type	File	Logic Contracts	Interfaces	Lines	nS
📝	contracts/reader/ContractReader.sol	1	__	76	72
📝	contracts/token/ShareToken.sol	1	__	21	21
📝	contracts/perpetual/Brokerage.sol	1	__	62	62
📝	contracts/perpetual/Position.sol	1	__	297	28
📝	contracts/perpetual/PerpetualGovernance.sol	1	__	102	10
📝	contracts/perpetual/Collateral.sol	1	__	159	15
📝	contracts/perpetual/Perpetual.sol	1	__	317	31
🔍	contracts/interface/IPerpetual.sol	__	1	71	10
🔍	contracts/interface/IPerpetualProxy.sol	__	1	85	18
🔍	contracts/interface/IPriceFeeder.sol	__	1	6	5
🔍	contracts/interface/IGlobalConfig.sol	__	1	7	4
🔍	contracts/interface/IChainlinkFeeder.sol	__	1	9	6
🔍	contracts/interface/IAMM.sol	__	1	44	9
📝	contracts/exchange/Exchange.sol	1	__	236	20
📝	contracts/proxy/PerpetualProxy.sol	1	__	157	15
📝	contracts/oracle/InversedChainlinkAdapter.sol	1	__	21	21
📝	contracts/oracle/ChainlinkAdapter.sol	1	__	20	20
📚	contracts/lib/LibSignature.sol	1	__	48	44
📚	contracts/lib/LibOrder.sol	1	__	143	13
📚	contracts/lib/LibEIP712.sol	1	__	30	30

Type	File	Logic Contracts	Interfaces	Lines	nS
📚	contracts/lib/LibTypes.sol	1	__	83	83
📚	contracts/lib/LibMath.sol	2	__	307	30
📝	contracts/global/GlobalConfig.sol	1	__	28	28
📝	contracts/liquidity/AMM.sol	1	__	782	73
📝	contracts/liquidity/AMMGovernance.sol	1	__	47	47
📝📚🔍	**Totals**	**20**	**6**	**3158**	**28**

Inline Documentation

- **Comment-to-Source Ratio:** On average there are `9.32` code lines per comment.
- **ToDo's:** `0`

Components

📝Contracts	📚Libraries	🔍Interfaces
14	6	6

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

🌐Public	💰Payable
212	11

External	Internal	Private	Pure	View
85	283	8	49	97

StateVariables

Total	🌐Public
58	30

Capabilities

Solidity Versions observed	🔦 Experimental Features	💰 Can Receive Funds	🖥️ Uses Assembly	💣 Has Destroyable Contracts
`^0.5.2`	`ABIEncoderV2`	`yes`	`yes` (1 asm blocks)	`no`

📤 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧮 Uses Hash Functions	🔖 ECRecover	🌀 New/Create/Create2
`yes`	`no`	`no`	`yes`	`yes`	`no`

A.2.2 MythX

MythX is a security analysis API for Ethereum smart contracts. It performs multiple types of analysis, including fuzzing and symbolic execution, to detect many common vulnerability types. The tool was used for automated vulnerability discovery for all audited contracts and libraries. More details on MythX can be found at mythx.io.

Below is the raw output of the MythX vulnerability scan:

```
Report for contracts/lib/LibSignature.sol
https://dashboard.mythx.io/#/console/analyses/6047d58d-1791-42fe-a5f1-8500f315dae3
╒══════╤═══════════════╤══════════╤═══════════════════════╕
│ Line │ SWC Title     │ Severity │ Short Description     │
╞══════╪═══════════════╪══════════╪═══════════════════════╡
│    1 │ Floating Pragma │ Low    │ A floating pragma is set. │
╘══════╧═══════════════╧══════════╧═══════════════════════╛

Report for contracts/exchange/Exchange.sol
https://dashboard.mythx.io/#/console/analyses/ee89f431-6db4-4971-9e67-dc3e159e956c
╒══════╤═══════════════════╤══════════╤═══════════════════════╕
│ Line │ SWC Title         │ Severity │ Short Description     │
╞══════╪═══════════════════╪══════════╪═══════════════════════╡
│    1 │ Floating Pragma   │ Low      │ A floating pragma is set. │
├──────┼───────────────────┼──────────┼───────────────────────┤
│  117 │ Requirement Violation │ Low  │ Requirement violation. │
├──────┼───────────────────┼──────────┼───────────────────────┤
│   11 │ Requirement Violation │ Low  │ Requirement violation. │
╘══════╧═══════════════════╧══════════╧═══════════════════════╛

Report for contracts/global/GlobalConfig.sol
https://dashboard.mythx.io/#/console/analyses/08afe18e-1328-44f3-b37f-e972afbf19b6
```

Line	SWC Title	Severity	Short Description
1	Floating Pragma	Low	A floating pragma is set.

Report for @openzeppelin/contracts/access/Roles.sol
https://dashboard.mythx.io/#/console/analyses/741aefc4-b04e-4dc6-a728-9543a769bb1d

Line	SWC Title	Severity	Short Description
1	Floating Pragma	Low	A floating pragma is set.

Report for contracts/test/TestPriceFeeder.sol
https://dashboard.mythx.io/#/console/analyses/1856d0ef-3be2-444e-ad54-43c14ac9bcf6

Line	SWC Title	Severity	Short Description
1	Floating Pragma	Low	A floating pragma is set.

Report for contracts/test/TestBrokerage.sol
https://dashboard.mythx.io/#/console/analyses/bc7e41d3-d310-4152-95ca-796d9c004d4f

Line	SWC Title	Severity	Short Description
1	Floating Pragma	Low	A floating pragma is set.
11	State Variable Default Visibility	Low	State variable visibility is not set.

Report for contracts/reader/ContractReader.sol
https://dashboard.mythx.io/#/console/analyses/da56e9d5-e0ac-4178-ba1d-81f2d4fd3cd5

Line	SWC Title	Severity	Short Description
1	Floating Pragma	Low	A floating pragma is set.
72	Requirement Violation	Low	Requirement violation.
9	Requirement Violation	Low	Requirement violation.

Report for contracts/perpetual/PerpetualGovernance.sol

https://dashboard.mythx.io/#/console/analyses/d8ead7bf-9c61-4374-afcc-06b180eee012

Line	SWC Title	Severity	Short Description
1	Floating Pragma	Low	A floating pragma is set.

Report for contracts/oracle/InversedChainlinkAdapter.sol
https://dashboard.mythx.io/#/console/analyses/8ac7c111-1436-4bd7-800b-c41ecc640df6

Line	SWC Title	Severity	Short Description
18	Integer Overflow and Underflow	High	The arithmetic operator can overflow.
1	Floating Pragma	Low	A floating pragma is set.
19	DoS with Failed Call	Low	Multiple calls are executed in the same transaction.

Report for @openzeppelin/contracts/token/ERC20/ERC20Burnable.sol
https://dashboard.mythx.io/#/console/analyses/5e47c8e8-a4ed-4870-9b3e-d2b1d69fff8f

Line	SWC Title	Severity	Short Description
1	Floating Pragma	Low	A floating pragma is set.

Report for contracts/test/TestPerpetualGovernance.sol
https://dashboard.mythx.io/#/console/analyses/e5ee874a-7004-46ca-b120-2555f750e60c

Line	SWC Title	Severity	Short Description
1	Floating Pragma	Low	A floating pragma is set.

Report for contracts/test/TestTypes.sol
https://dashboard.mythx.io/#/console/analyses/73f7ec4a-14eb-45df-bb92-82d724ed2aa5

Line	SWC Title	Severity	Short Description
1	Floating Pragma	Low	A floating pragma is set.

Report for contracts/test/TestFundingMock.sol
https://dashboard.mythx.io/#/console/analyses/b75e455a-0568-47fa-8301-9e9d79425f49

Line	SWC Title	Severity	Short Description

Line	SWC Title	Severity	Short Description
1	Floating Pragma	Low	A floating pragma is set.
36	State Variable Default Visibility	Low	State variable visibility is not set.

Report for contracts/test/TestAMM.sol
https://dashboard.mythx.io/#/console/analyses/77de240b-0a47-47f6-b814-70af1135bf34

Line	SWC Title	Severity	Short Description
1	Floating Pragma	Low	A floating pragma is set.

Report for contracts/lib/LibOrder.sol
https://dashboard.mythx.io/#/console/analyses/7729c6f0-6e9e-4890-aa73-e9e3bc506269

Line	SWC Title	Severity	Short Description
1	Floating Pragma	Low	A floating pragma is set.

Report for contracts/test/TestToken.sol
https://dashboard.mythx.io/#/console/analyses/7e2d60b2-1772-4025-b519-dcc0bec73da0

Line	SWC Title	Severity	Short Description
49	Unknown	Medium	Incorrect ERC20 implementation
39	Assert Violation	Medium	An assertion violation was triggered.
1	Floating Pragma	Low	A floating pragma is set.
52	State Variable Default Visibility	Low	State variable visibility is not set.
33	Assert Violation	Low	An assertion violation was triggered.

```
|    192 | Shadowing State Variables          | Low        | State variable shadows
another state variable. |
╚════════╧════════════════════════════════════╧════════════╧═══════════════════════
```

Report for contracts/test/TestSignature.sol
https://dashboard.mythx.io/#/console/analyses/74462b47-a65f-4403-9925-3c6c3ccbe197

```
╒════════╤═══════════════════════════╤════════════╤═══════════════════════════════╕
│   Line │ SWC Title                 │ Severity   │ Short Description             │
╞════════╪═══════════════════════════╪════════════╪═══════════════════════════════╡
│      1 │ Floating Pragma           │ Low        │ A floating pragma is set.     │
╘════════╧═══════════════════════════╧════════════╧═══════════════════════════════╛
```

Report for contracts/lib/LibMath.sol
https://dashboard.mythx.io/#/console/analyses/55066a9a-2c85-494d-8ba7-f0adfa8ef5f5

```
╒════════╤═══════════════════════════════╤════════════╤═══════════════════════════
│   Line │ SWC Title                     │ Severity   │ Short Description
│
╞════════╪═══════════════════════════════╪════════════╪═══════════════════════════

│      1 │ Floating Pragma               │ Low        │ A floating pragma is set.
│
├────────┼───────────────────────────────┼────────────┼──────────────────────────

│    109 │ DoS With Block Gas Limit      │ Low        │ Loop over unbounded data
structure. │
├────────┼───────────────────────────────┼────────────┼──────────────────────────

│    168 │ DoS With Block Gas Limit      │ Low        │ Loop over unbounded data
structure. │
├────────┼───────────────────────────────┼────────────┼──────────────────────────

│    172 │ DoS With Block Gas Limit      │ Low        │ Loop over unbounded data
structure. │
╘════════╧═══════════════════════════════╧════════════╧═══════════════════════════
```

Report for @openzeppelin/contracts/math/SafeMath.sol
https://dashboard.mythx.io/#/console/analyses/f8da3ca7-62e2-4353-83e9-1ca5f720b9a1

```
╒════════╤═══════════════════════════╤════════════╤═══════════════════════════════╕
│   Line │ SWC Title                 │ Severity   │ Short Description             │
╞════════╪═══════════════════════════╪════════════╪═══════════════════════════════╡
│      1 │ Floating Pragma           │ Low        │ A floating pragma is set.     │
╘════════╧═══════════════════════════╧════════════╧═══════════════════════════════╛
```

Report for contracts/token/ShareToken.sol
https://dashboard.mythx.io/#/console/analyses/f915729c-0df1-4396-8f4c-e7639d48aea9

```
╒════════╤═══════════════════════════╤════════════╤═══════════════════════════════╕
│   Line │ SWC Title                 │ Severity   │ Short Description             │
╞════════╪═══════════════════════════╪════════════╪═══════════════════════════════╡
│      1 │ Floating Pragma           │ Low        │ A floating pragma is set.     │
╘════════╧═══════════════════════════╧════════════╧═══════════════════════════════╛
```

Report for contracts/perpetual/Perpetual.sol
https://dashboard.mythx.io/#/console/analyses/a909e39b-afcd-47c3-b4dc-d2d914e3d82c

Line	SWC Title	Severity	Short Description
1	Floating Pragma	Low	A floating pragma is set.

Report for contracts/oracle/ChainlinkAdapter.sol
https://dashboard.mythx.io/#/console/analyses/d7c7a24f-593b-4ce0-8d1f-97748904c849

Line	SWC Title	Severity	Short Description
1	Floating Pragma	Low	A floating pragma is set.

Report for contracts/test/TestMath.sol
https://dashboard.mythx.io/#/console/analyses/cab62764-db1e-4368-850a-b29946a73d17

Line	SWC Title	Severity	Short Description
1	Floating Pragma	Low	A floating pragma is set.

Report for contracts/test/TestToken.sol
https://dashboard.mythx.io/#/console/analyses/de774a6c-178f-49ea-b1f3-ab25199da540

Line	SWC Title	Severity	Short Description
49	Unknown	Medium	Incorrect ERC20 implementation
1	Floating Pragma	Low	A floating pragma is set.
52	State Variable Default Visibility	Low	State variable visibility is not set.
192	Shadowing State Variables	Low	State variable shadows another state variable.

Report for @openzeppelin/contracts/token/ERC20/ERC20Mintable.sol
https://dashboard.mythx.io/#/console/analyses/ae378fc1-dbf7-4c10-b0f9-5a0bc3429010

Line	SWC Title	Severity	Short Description
1	Floating Pragma	Low	A floating pragma is set.

Report for contracts/perpetual/Position.sol
https://dashboard.mythx.io/#/console/analyses/96a418fe-69fe-4dda-aa61-2a19d233687e

```
╒══════╤═══════════════╤════════════╤═══════════════════╕
│ Line │ SWC Title     │ Severity   │ Short Description │
╞══════╪═══════════════╪════════════╪═══════════════════╡
│    1 │ Floating Pragma │ Low      │ A floating pragma is set. │
╘══════╧═══════════════╧════════════╧═══════════════════╛
```

Report for @openzeppelin/contracts/token/ERC20/SafeERC20.sol
https://dashboard.mythx.io/#/console/analyses/9b342c42-3ae8-4f6b-b169-7244c353ec11

```
╒══════╤═══════════════╤════════════╤═══════════════════╕
│ Line │ SWC Title     │ Severity   │ Short Description │
╞══════╪═══════════════╪════════════╪═══════════════════╡
│    1 │ Floating Pragma │ Low      │ A floating pragma is set. │
╘══════╧═══════════════╧════════════╧═══════════════════╛
```

Report for @openzeppelin/contracts/utils/Address.sol
https://dashboard.mythx.io/#/console/analyses/517d5d70-94db-4da3-93a7-07f2317d4d4e

```
╒══════╤═══════════════╤════════════╤═══════════════════╕
│ Line │ SWC Title     │ Severity   │ Short Description │
╞══════╪═══════════════╪════════════╪═══════════════════╡
│    1 │ Floating Pragma │ Low      │ A floating pragma is set. │
╘══════╧═══════════════╧════════════╧═══════════════════╛
```

Report for contracts/lib/LibEIP712.sol
https://dashboard.mythx.io/#/console/analyses/e6301b3d-22ff-4a8d-8791-4e27dac38fab

```
╒══════╤═══════════════════════╤════════════╤═══════════════════╕
│ Line │ SWC Title             │ Severity   │ Short Description │
╞══════╪═══════════════════════╪════════════╪═══════════════════╡
│   27 │ Unknown               │ Medium     │ Incorrect function "hashEIP712Message" state mutability │
├──────┼───────────────────────┼────────────┼───────────────────┤
│    1 │ Floating Pragma       │ Low        │ A floating pragma is set. │
├──────┼───────────────────────┼────────────┼───────────────────┤
│   18 │ DoS With Block Gas Limit │ Low     │ Potentially unbounded data structure passed to builtin. │
╘══════╧═══════════════════════╧════════════╧═══════════════════╛
```

Report for contracts/test/TestToken.sol
https://dashboard.mythx.io/#/console/analyses/fc1ab4bb-f661-4f85-8349-ae3c98eb41c5

```
╒══════╤═══════════════════════╤════════════╤═══════════════════╕
│ Line │ SWC Title             │ Severity   │ Short Description │
╞══════╪═══════════════════════╪════════════╪═══════════════════╡
│   49 │ Unknown               │ Medium     │ Incorrect ERC20 implementation │
├──────┼───────────────────────┼────────────┼───────────────────┤
│    1 │ Floating Pragma       │ Low        │ A floating pragma is set.
```

```
│
├──────────┼────────────────────────────────────────┼──────────────┼──────────────

│     52 │ State Variable Default Visibility │ Low        │ State variable visibility
is not set.          │
├──────────┼────────────────────────────────────────┼──────────────┼──────────────

│    192 │ Shadowing State Variables         │ Low        │ State variable shadows
another state variable. │
╘══════════╧════════════════════════════════════════╧══════════════╧══════════════

Report for contracts/liquidity/AMM.sol
https://dashboard.mythx.io/#/console/analyses/25349e5a-3f27-4e06-905e-f8b41ff66599
╒════════╤════════════════╤════════════╤═════════════════════════╕
│   Line │ SWC Title      │ Severity   │ Short Description       │
╞════════╪════════════════╪════════════╪═════════════════════════╡
│      1 │ Floating Pragma │ Low        │ A floating pragma is set. │
╘════════╧════════════════╧════════════╧═════════════════════════╛
Report for contracts/test/TestOrder.sol
https://dashboard.mythx.io/#/console/analyses/1d33a597-5dd9-4118-85c6-cd7808496b3e
╒════════╤════════════════╤════════════╤═════════════════════════╕
│   Line │ SWC Title      │ Severity   │ Short Description       │
╞════════╪════════════════╪════════════╪═════════════════════════╡
│      1 │ Floating Pragma │ Low        │ A floating pragma is set. │
╘════════╧════════════════╧════════════╧═════════════════════════╛
Report for contracts/test/TestPosition.sol
https://dashboard.mythx.io/#/console/analyses/742d22bd-0e4c-4f3d-939a-f1d07a56297d
╒════════╤════════════════╤════════════╤═════════════════════════╕
│   Line │ SWC Title      │ Severity   │ Short Description       │
╞════════╪════════════════╪════════════╪═════════════════════════╡
│      1 │ Floating Pragma │ Low        │ A floating pragma is set. │
╘════════╧════════════════╧════════════╧═════════════════════════╛
Report for contracts/perpetual/Brokerage.sol
https://dashboard.mythx.io/#/console/analyses/1f0a8138-1b11-4fc9-abc2-916c3c723aec
╒══════════╤═══════════════════════════════════════════════════════════╤══════════════

│   Line │ SWC Title                                        │ Severity   │ Short
Description                                                             │
╞══════════╪═══════════════════════════════════════════════════════════╪══════════════

│      1 │ Floating Pragma                                  │ Low        │ A floating
pragma is set.                                                  │
├──────────┼───────────────────────────────────────────────────────────┼──────────────

│     30 │ Weak Sources of Randomness from Chain Attributes │ Low        │ Potential
use of "block.number" as source of randomness.                           │
├──────────┼───────────────────────────────────────────────────────────┼──────────────

│     32 │ Weak Sources of Randomness from Chain Attributes │ Low        │ Potential
use of "block.number" as source of randomness.                           │
├──────────┼───────────────────────────────────────────────────────────┼──────────────
```

```
|     45 | Weak Sources of Randomness from Chain Attributes | Low        | Potential
use of "block.number" as source of randonmness.                          |
|--------|--------------------------------------------------|------------|-----------

|     56 | Weak Sources of Randomness from Chain Attributes | Low        | Potential
use of "block.number" as source of randonmness.                          |
|--------|--------------------------------------------------|------------|-----------

|     56 | Weak Sources of Randomness from Chain Attributes | Low        | A control
flow decision is made based on The block.number environment variable. |
|--------|--------------------------------------------------|------------|-----------

Report for contracts/perpetual/Collateral.sol
https://dashboard.mythx.io/#/console/analyses/21119bf4-8cd7-4beb-9c76-fdb972f25835
|--------|--------------------------------------------------|------------|-----------

|   Line | SWC Title                                        | Severity   | Short
Description                                  |
|--------|--------------------------------------------------|------------|-----------

|      1 | Floating Pragma                                  | Low        | A floating
pragma is set.                               |
|--------|--------------------------------------------------|------------|-----------

|     62 | Weak Sources of Randomness from Chain Attributes | Low        | Potential
use of "block.number" as source of randonmness. |
|--------|--------------------------------------------------|------------|-----------

|     71 | Weak Sources of Randomness from Chain Attributes | Low        | Potential
use of "block.number" as source of randonmness. |
|--------|--------------------------------------------------|------------|-----------

Report for @openzeppelin/contracts/access/roles/WhitelistedRole.sol
https://dashboard.mythx.io/#/console/analyses/ef339987-71f9-4bc3-886f-e70d937b1728
|--------|-----------------|------------|-----------------------------|
|   Line | SWC Title       | Severity   | Short Description           |
|--------|-----------------|------------|-----------------------------|
|      1 | Floating Pragma | Low        | A floating pragma is set.   |
|--------|-----------------|------------|-----------------------------|
Report for contracts/Migrations.sol
https://dashboard.mythx.io/#/console/analyses/11d9299f-940a-4949-a87c-d7be1911681c
|--------|-----------------|------------|-----------------------------|

|   Line | SWC Title       | Severity   | Short Description
|
|--------|-----------------|------------|-----------------------------|

|      1 | Floating Pragma | Low        | A floating pragma is set.
|
|--------|-----------------|------------|-----------------------------|

|     21 | Reentrancy      | Low        | A call to a user-supplied address is
```

```
executed. │
╘════════╧═══════════════════╧═════════════╧═══════════════════

Report for contracts/test/TestMath.sol
https://dashboard.mythx.io/#/console/analyses/c6fc0bc1-f65f-4785-accb-6d1c27e14e51
╒════════╤════════════════╤════════════╤═══════════════════════════╕
│   Line │ SWC Title      │ Severity   │ Short Description         │
╞════════╪════════════════╪════════════╪═══════════════════════════╡
│      1 │ Floating Pragma │ Low        │ A floating pragma is set. │
╘════════╧════════════════╧════════════╧═══════════════════════════╛
Report for contracts/lib/LibMath.sol
https://dashboard.mythx.io/#/console/analyses/223038b8-f6a7-4502-a3c1-3e265bb7c80c
╒════════╤══════════════════════════╤════════════╤═══════════════════
│   Line │ SWC Title                │ Severity   │ Short Description
│
╞════════╪══════════════════════════╪════════════╪═══════════════════
│      1 │ Floating Pragma          │ Low        │ A floating pragma is set.
│
├────────┼──────────────────────────┼────────────┼───────────────────
│    109 │ DoS With Block Gas Limit │ Low        │ Loop over unbounded data
structure. │
├────────┼──────────────────────────┼────────────┼───────────────────
│    168 │ DoS With Block Gas Limit │ Low        │ Loop over unbounded data
structure. │
├────────┼──────────────────────────┼────────────┼───────────────────
│    172 │ DoS With Block Gas Limit │ Low        │ Loop over unbounded data
structure. │
╘════════╧══════════════════════════╧════════════╧═══════════════════

Report for contracts/liquidity/AMMGovernance.sol
https://dashboard.mythx.io/#/console/analyses/7a5f404c-3296-4860-a630-2d039e156e46
╒════════╤════════════════╤════════════╤═══════════════════════════╕
│   Line │ SWC Title      │ Severity   │ Short Description         │
╞════════╪════════════════╪════════════╪═══════════════════════════╡
│      1 │ Floating Pragma │ Low        │ A floating pragma is set. │
╘════════╧════════════════╧════════════╧═══════════════════════════╛
Report for contracts/lib/LibTypes.sol
https://dashboard.mythx.io/#/console/analyses/8f7324fd-e740-4cd2-ab14-1d8c80bbc081
╒════════╤════════════════╤════════════╤═══════════════════════════╕
│   Line │ SWC Title      │ Severity   │ Short Description         │
╞════════╪════════════════╪════════════╪═══════════════════════════╡
│      1 │ Floating Pragma │ Low        │ A floating pragma is set. │
╘════════╧════════════════╧════════════╧═══════════════════════════╛
Report for @openzeppelin/contracts/token/ERC20/ERC20.sol
https://dashboard.mythx.io/#/console/analyses/99f431e4-129d-491a-a1a5-1ef37e47b9af
╒════════╤════════════════╤════════════╤═══════════════════════════╕
│   Line │ SWC Title      │ Severity   │ Short Description         │
```

```
├────────┼──────────────────┼────────────┼────────────────────────────┤
│      1 │ Floating Pragma  │ Low        │ A floating pragma is set.  │
└────────┴──────────────────┴────────────┴────────────────────────────┘
Report for contracts/test/TestCollateral.sol
https://dashboard.mythx.io/#/console/analyses/0f34b74d-c3f0-4cb0-a44d-957052a0d8b1
┌────────┬──────────────────┬────────────┬────────────────────────────┐
│   Line │ SWC Title        │ Severity   │ Short Description          │
╞════════╪══════════════════╪════════════╪════════════════════════════╡
│      1 │ Floating Pragma  │ Low        │ A floating pragma is set.  │
└────────┴──────────────────┴────────────┴────────────────────────────┘
Report for contracts/test/TestPerpetual.sol
https://dashboard.mythx.io/#/console/analyses/0a61dc46-295a-4126-938a-ee8fc428dffb
┌────────┬──────────────────┬────────────┬────────────────────────────┐
│   Line │ SWC Title        │ Severity   │ Short Description          │
╞════════╪══════════════════╪════════════╪════════════════════════════╡
│      1 │ Floating Pragma  │ Low        │ A floating pragma is set.  │
└────────┴──────────────────┴────────────┴────────────────────────────┘
Report for contracts/proxy/PerpetualProxy.sol
https://dashboard.mythx.io/#/console/analyses/3a0d1dec-abbb-4dce-b395-4f59411832bc
┌────────┬────────────────────────────────────┬────────────┬─────────
│   Line │ SWC Title                          │ Severity   │ Short Description
│
╞════════╪════════════════════════════════════╪════════════╪═════════
│      1 │ Floating Pragma                    │ Low        │ A floating pragma is set.
│
├────────┼────────────────────────────────────┼────────────┼─────────
│     13 │ State Variable Default Visibility │ Low        │ State variable visibility
is not set. │
└────────┴────────────────────────────────────┴────────────┴─────────
```

A.2.3 Surya

Surya is a utility tool for smart contract systems. It provides a number of visual outputs and information about the structure of smart contracts. It also supports querying the function call graph in multiple ways to aid in the manual inspection and control flow analysis of contracts.

Below is a complete list of functions with their visibility and modifiers:

Contract	Type	Bases	
└	**Function Name**	**Visibility**	**Mut**
ShareToken	Implementation	ERC20Mintable	
└		Public ❗️	🛑
└	burn	Public ❗️	🛑

Contract	Type	Bases	
ContractReader	Implementation		
└	getGovParams	Public ❗	
└	getPerpetualStorage	Public ❗	
└	getAccountStorage	Public ❗	
Brokerage	Implementation		
└	setBroker	Internal 🔒	
└	currentBroker	Public ❗	
└	getBroker	Public ❗	
Position	Implementation	Collateral, PerpetualGovernance	
└		Public ❗	
└	socialLossPerContract	Public ❗	
└	totalSize	Public ❗	
└	getPosition	Public ❗	
└	calculateLiquidateAmount	Public ❗	
└	addSocialLossPerContract	Internal 🔒	
└	marginBalanceWithPrice	Internal 🔒	
└	availableMarginWithPrice	Internal 🔒	
└	marginWithPrice	Internal 🔒	
└	maintenanceMarginWithPrice	Internal 🔒	
└	drawableBalanceWithPrice	Internal 🔒	
└	pnlWithPrice	Internal 🔒	
└	increaseTotalSize	Internal 🔒	
└	decreaseTotalSize	Internal 🔒	
└	socialLoss	Internal 🔒	
└	socialLossWithAmount	Internal 🔒	

Contract	Type	Bases	
└	fundingLoss	Internal 🔒	
└	fundingLossWithAmount	Internal 🔒	
└	remargin	Internal 🔒	
└	calculatePnl	Internal 🔒	
└	open	Internal 🔒	
└	close	Internal 🔒	
└	trade	Internal 🔒	
└	handleSocialLoss	Internal 🔒	
└	liquidate	Internal 🔒	
PerpetualGovernance	Implementation	WhitelistedRole	
└	getGovernance	Public ❗	
└	setGovernanceParameter	Public ❗	
└	setGovernanceAddress	Public ❗	
└	beginGlobalSettlement	Public ❗	
Collateral	Implementation		
└		Public ❗	
└	getCashBalance	Public ❗	
└	isTokenizedCollateral	Internal 🔒	
└	deposit	Internal 🔒	
└	applyForWithdrawal	Internal 🔒	
└	_withdraw	Private 🔐	
└	withdraw	Internal 🔒	
└	depositToProtocol	Internal 🔒	
└	withdrawFromProtocol	Internal 🔒	
└	withdrawAll	Internal 🔒	
└	updateBalance	Internal 🔒	

Contract	Type	Bases	
└	ensurePositiveBalance	Internal 🔒	
└	transferBalance	Internal 🔒	
└	toWad	Internal 🔒	
└	toCollateral	Internal 🔒	
Perpetual	Implementation	Brokerage, Position	
└		Public ❗	
└	setCashBalance	Public ❗	
└		External ❗	
└	markPrice	Public ❗	
└	setBroker	Public ❗	
└	setBrokerFor	Public ❗	
└	depositToAccount	Private 🔐	
└	depositFor	Public ❗	
└	depositEtherFor	Public ❗	
└	deposit	Public ❗	
└	depositEther	Public ❗	
└	depositAndSetBroker	Public ❗	
└	depositEtherAndSetBroker	Public ❗	
└	applyForWithdrawal	Public ❗	
└	settleFor	Private 🔐	
└	settle	Public ❗	
└	endGlobalSettlement	Public ❗	
└	withdrawFromAccount	Private 🔐	
└	withdrawFor	Public ❗	
└	withdraw	Public ❗	
└	depositToInsuranceFund	Public ❗	

Contract	Type	Bases	
└	depositEtherToInsuranceFund	Public ❗	
└	withdrawFromInsuranceFund	Public ❗	
└	positionMargin	Public ❗	
└	maintenanceMargin	Public ❗	
└	marginBalance	Public ❗	
└	pnl	Public ❗	
└	availableMargin	Public ❗	
└	drawableBalance	Public ❗	
└	isSafe	Public ❗	
└	isSafeWithPrice	Public ❗	
└	isBankrupt	Public ❗	
└	isIMSafe	Public ❗	
└	isIMSafeWithPrice	Public ❗	
└	liquidateFrom	Public ❗	
└	liquidate	Public ❗	
└	tradePosition	Public ❗	
└	transferCashBalance	Public ❗	
IPerpetual	Interface		
└	devAddress	External ❗	
└	getCashBalance	External ❗	
└	getPosition	External ❗	
└	getBroker	External ❗	
└	getGovernance	External ❗	
└	status	External ❗	
└	settlementPrice	External ❗	
└	globalConfig	External ❗	

Contract	Type	Bases	
└	collateral	External ❗	
└	isWhitelisted	External ❗	
└	currentBroker	External ❗	
└	amm	External ❗	
└	totalSize	External ❗	
└	markPrice	External ❗	
└	socialLossPerContract	External ❗	
└	availableMargin	External ❗	
└	positionMargin	External ❗	
└	maintenanceMargin	External ❗	
└	isSafe	External ❗	
└	isSafeWithPrice	External ❗	
└	isIMSafe	External ❗	
└	isIMSafeWithPrice	External ❗	
└	tradePosition	External ❗	
└	transferCashBalance	External ❗	
└	setBrokerFor	External ❗	
└	depositFor	External ❗	
└	depositEtherFor	External ❗	
└	withdrawFor	External ❗	
└	liquidate	External ❗	
└	liquidateFrom	External ❗	
└	insuranceFundBalance	External ❗	
IPerpetualProxy	Interface		
└	self	External ❗	
└	perpetual	External ❗	

Contract	Type	Bases	
└	devAddress	External ❗	
└	currentBroker	External ❗	
└	markPrice	External ❗	
└	settlementPrice	External ❗	
└	availableMargin	External ❗	
└	getPoolAccount	External ❗	
└	cashBalance	External ❗	
└	positionSize	External ❗	
└	positionSide	External ❗	
└	positionEntryValue	External ❗	
└	positionEntrySocialLoss	External ❗	
└	positionEntryFundingLoss	External ❗	
└	status	External ❗	
└	socialLossPerContract	External ❗	
└	transferBalanceIn	External ❗	
└	transferBalanceOut	External ❗	
└	transferBalanceTo	External ❗	
└	trade	External ❗	
└	setBrokerFor	External ❗	
└	depositFor	External ❗	
└	depositEtherFor	External ❗	
└	withdrawFor	External ❗	
└	isSafe	External ❗	
└	isSafeWithPrice	External ❗	
└	isProxySafe	External ❗	
└	isProxySafeWithPrice	External ❗	
└	isIMSafe	External ❗	

Contract	Type	Bases	
└	isIMSafeWithPrice	External ❗	
└	lotSize	External ❗	
└	tradingLotSize	External ❗	
IPriceFeeder	Interface		
└	price	External ❗	
IGlobalConfig	Interface		
└	withdrawalLockBlockCount	External ❗	
└	brokerLockBlockCount	External ❗	
IChainlinkFeeder	Interface		
└	latestAnswer	External ❗	
└	latestTimestamp	External ❗	
IAMM	Interface		
└	shareTokenAddress	External ❗	
└	lastFundingState	External ❗	
└	getGovernance	External ❗	
└	perpetualProxy	External ❗	
└	currentMarkPrice	External ❗	
└	currentAvailableMargin	External ❗	
└	currentFairPrice	External ❗	
└	positionSize	External ❗	
└	currentAccumulatedFundingPerContract	External ❗	
└	settleShare	External ❗	
└	buy	External ❗	
└	sell	External ❗	
└	buyFromWhitelisted	External ❗	

Contract	Type	Bases	
└	sellFromWhitelisted	External ❗	
└	buyFrom	External ❗	
└	sellFrom	External ❗	
Exchange	Implementation		
└	matchOrders	Public ❗	
└	fillOrder	Internal 🔒	
└	matchOrderWithAMM	Public ❗	
└	validatePrice	Internal 🔒	
└	validateOrderParam	Internal 🔒	
└	claimTradingFee	Internal 🔒	
└	cancelOrder	Public ❗	
└	claimDevFee	Internal 🔒	
└	claimTakerDevFee	Internal 🔒	
└	claimMakerDevFee	Internal 🔒	
PerpetualProxy	Implementation		
└		Public ❗	
└	self	Public ❗	
└	status	Public ❗	
└	devAddress	Public ❗	
└	markPrice	Public ❗	
└	settlementPrice	Public ❗	
└	currentBroker	Public ❗	
└	availableMargin	Public ❗	
└	getPoolAccount	Public ❗	
└	cashBalance	Public ❗	
└	positionSize	Public ❗	

Contract	Type	Bases	
└	positionSide	Public ❗	
└	positionEntryValue	Public ❗	
└	positionEntrySocialLoss	Public ❗	
└	positionEntryFundingLoss	Public ❗	
└	socialLossPerContract	Public ❗	
└	transferBalanceIn	Public ❗	
└	transferBalanceOut	Public ❗	
└	transferBalanceTo	Public ❗	
└	trade	Public ❗	
└	setBrokerFor	Public ❗	
└	depositFor	Public ❗	
└	depositEtherFor	Public ❗	
└	withdrawFor	Public ❗	
└	isSafe	Public ❗	
└	isSafeWithPrice	Public ❗	
└	isProxySafe	Public ❗	
└	isProxySafeWithPrice	Public ❗	
└	isIMSafe	Public ❗	
└	isIMSafeWithPrice	Public ❗	
└	lotSize	Public ❗	
└	tradingLotSize	Public ❗	
InversedChainlinkAdapter	Implementation		
└		Public ❗	
└	price	Public ❗	
ChainlinkAdapter	Implementation		
└		Public ❗	

Contract	Type	Bases	
└	price	Public ❗	
LibSignature	Library		
└	isValidSignature	Internal 🔒	
LibOrder	Library		
└	getOrderHash	Internal 🔒	
└	getOrderHash	Internal 🔒	
└	getOrder	Internal 🔒	
└	hashOrder	Internal 🔒	
└	getOrderVersion	Internal 🔒	
└	getExpiredAt	Internal 🔒	
└	isSell	Internal 🔒	
└	getPrice	Internal 🔒	
└	isMarketOrder	Internal 🔒	
└	isMarketBuy	Internal 🔒	
└	isMakerOnly	Internal 🔒	
└	isInversed	Internal 🔒	
└	side	Internal 🔒	
└	makerFeeRate	Internal 🔒	
└	takerFeeRate	Internal 🔒	
LibEIP712	Library		
└	hashEIP712Message	Internal 🔒	
LibTypes	Library		
└	counterSide	Internal 🔒	
LibMathSigned	Library		
└	WAD	Internal 🔒	

Contract	Type	Bases	
└	neg	Internal 🔒	
└	mul	Internal 🔒	
└	div	Internal 🔒	
└	sub	Internal 🔒	
└	add	Internal 🔒	
└	wmul	Internal 🔒	
└	wdiv	Internal 🔒	
└	wfrac	Internal 🔒	
└	min	Internal 🔒	
└	max	Internal 🔒	
└	toUint256	Internal 🔒	
└	wpowi	Internal 🔒	
└	roundHalfUp	Internal 🔒	
└	wln	Internal 🔒	
└	logBase	Internal 🔒	
└	ceil	Internal 🔒	
LibMathUnsigned	Library		
└	WAD	Internal 🔒	
└	add	Internal 🔒	
└	sub	Internal 🔒	
└	mul	Internal 🔒	
└	div	Internal 🔒	
└	wmul	Internal 🔒	
└	wdiv	Internal 🔒	
└	wfrac	Internal 🔒	
└	min	Internal 🔒	

Contract	Type	Bases	
└	max	Internal 🔒	
└	toInt256	Internal 🔒	
└	mod	Internal 🔒	
└	ceil	Internal 🔒	
GlobalConfig	Implementation	WhitelistedRole	
└		Public ❗	
└	setGlobalParameter	Public ❗	
AMM	Implementation	AMMGovernance	
└		Public ❗	
└	authorizedBroker	Internal 🔒	
└	shareTokenAddress	Public ❗	
└	indexPrice	Public ❗	
└	positionSize	Public ❗	
└	lastFundingState	Public ❗	
└	lastAvailableMargin	Internal 🔒	
└	lastFairPrice	Internal 🔒	
└	lastPremium	Internal 🔒	
└	lastEMAPremium	Internal 🔒	
└	lastMarkPrice	Internal 🔒	
└	lastPremiumRate	Internal 🔒	
└	lastFundingRate	Public ❗	
└	currentFundingState	Public ❗	
└	currentAvailableMargin	Public ❗	
└	currentFairPrice	Public ❗	
└	currentPremium	Public ❗	
└	currentMarkPrice	Public ❗	

Contract	Type	Bases	
└	currentPremiumRate	Public ❗	
└	currentFundingRate	Public ❗	
└	currentAccumulatedFundingPerContract	Public ❗	
└	createPool	Public ❗	
└	getBuyPrice	Internal 🔒	
└	buyFrom	Private 🔐	
└	buyFromWhitelisted	Public ❗	
└	buy	Public ❗	
└	getSellPrice	Internal 🔒	
└	sellFrom	Private 🔐	
└	sellFromWhitelisted	Public ❗	
└	sell	Public ❗	
└	addLiquidity	Public ❗	
└	removeLiquidity	Public ❗	
└	settleShare	Public ❗	
└	depositAndBuy	Public ❗	
└	depositEtherAndBuy	Public ❗	
└	depositAndSell	Public ❗	
└	depositEtherAndSell	Public ❗	
└	buyAndWithdraw	Public ❗	
└	sellAndWithdraw	Public ❗	
└	depositAndAddLiquidity	Public ❗	
└	depositEtherAndAddLiquidity	Public ❗	
└	updateIndex	Public ❗	
└	initFunding	Private 🔐	
└	funding	Public ❗	
└	getBlockTimestamp	Internal 🔒	

Contract	Type	Bases	
└	currentXY	Internal 🔒	
└	availableMarginFromPoolAccount	Internal 🔒	
└	fairPriceFromPoolAccount	Internal 🔒	
└	premiumFromPoolAccount	Internal 🔒	
└	mustSafe	Internal 🔒	
└	mintShareTokenTo	Internal 🔒	
└	burnShareTokenFrom	Internal 🔒	
└	forceFunding	Internal 🔒	
└	forceFunding	Internal 🔒	
└	nextStateWithTimespan	Private 🔐	
└	timeOnFundingCurve	Internal 🔒	
└	integrateOnFundingCurve	Internal 🔒	
└	getAccumulatedFunding	Internal 🔒	
AMMGovernance	Implementation	WhitelistedRole	
└	setGovernanceParameter	Public ❗	
└	getGovernance	Public ❗	

Legend

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

A.2.4 Tests Suite

Below is the output generated by running the test suite:

```
Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.
```

```
Contract: AccumulatedFunding
  ✓ alpha2 (46ms)
  ✓ timeOnFundingCurve - upward (406ms)
  ✓ timeOnFundingCurve - critical (379ms)
  ✓ integrateOnFundingCurve - upward (317ms)
  ✓ integrateOnFundingCurve - downward (348ms)
  ✓ getAccumulatedFunding (20888ms)

Contract: amm
  exceptions
    ✓ indexPrice (1720ms)
    ✓ invalid broker (111ms)
    ✓ empty pool (1161ms)
    ✓ empty pool (948ms)
    ✓ empty pool (818ms)
    ✓ wrong perpetual status (82ms)
  composed interface
    ✓ depositAndBuy (208ms)
    ✓ depositAndSell (321ms)
  availableMargin
    ✓ without loss (182ms)
    ✓ loss is increasing (281ms)
  create amm
    ✓ spend: no marginBalance (1957ms)
    ✓ should success (4680ms)
    ✓ duplicated (1573ms)
  trading
    ✓ removeLiquidity - no position on removing liqudity (5237ms)
    ✓ removeLiquidity - transfer share (3896ms)
    ✓ removeLiquidity - success (3173ms)
    ✓ buy - success (2568ms)
    ✓ buyAndWithdraw - success (2909ms)
    ✓ buyAndWithdraw - success (524ms)
    ✓ buy - fail - price limit (2283ms)
    ✓ buy - success - pnl < 0, critical deposit amount (1734ms)
    ✓ buy withdraw 0 (1744ms)
    ✓ buy - withdraw (2532ms)
    ✓ buy - fail - pnl < 0, lower than critical deposit amount (2687ms)
    ✓ buy - fail - deadline (1846ms)
    ✓ sell - fail - price unsafe (2512ms)
    ✓ sell - fail - price limit (2126ms)
    ✓ sell - success (2409ms)
    ✓ buy and sell - success (5084ms)
    ✓ sell - success - large amount (2443ms)
    ✓ sell - fail - deadline (1533ms)
    ✓ addLiquidity - fail - no marginBalance (2204ms)
    ✓ addLiquidity - fail - unsafe (2424ms)
    ✓ addLiquidity - success (1739ms)
    ✓ depositAndAddLiquidity - success (1821ms)
    ✓ removeLiquidity - fail - shareBalance limited (493ms)
    ✓ updateIndex (1032ms)
```

```
    funding
      ✓ user buys => price increases (above the limit) => long position pays for
fundingLoss (7441ms)
      ✓ user buys => price increases (below the limit) => long position pays for
fundingLoss (6354ms)
estimateGas 15s: 113970
estimateGas 10m: 130876
estimateGas 1d: 154448
estimateGas 1y: 172181
      ✓ consumed gas (2865ms)
    composite helper
      ✓ depositAndBuy - success (2676ms)
      ✓ depositAndBuy, deposit = $0 - success (2092ms)
      ✓ depositAndSell - success (2664ms)
      ✓ depositAndAddLiquidity - success (1918ms)
    inverse contract
      ✓ depositAndBuy - success (1880ms)
      ✓ buyAndWithdraw - success (3816ms)

  Contract: amm-zero-cash
    trading
      ✓ updateIndex - fail - dev account is empty (1278ms)
      ✓ buy - success - without cash (6822ms)
      ✓ addLiquidity - success - using pnl (9649ms)

  Contract: amm-eth
    composite helper
      ✓ depositAndBuy - success (2833ms)
      ✓ depositAndBuy, deposit = $0 - success (1917ms)
      ✓ depositAndSell - success (2601ms)
      ✓ depositAndAddLiquidity - success (2040ms)
    create amm
      ✓ should success (4476ms)
      ✓ duplicated (1506ms)
    trading
      ✓ buy - success (2660ms)
      ✓ buy - fail - price limit (2295ms)
      ✓ buy - success - pnl < 0, critical deposit amount (1743ms)
      ✓ buy - fail - pnl < 0, lower than critical deposit amount (2827ms)
      ✓ buy - fail - deadline (1490ms)
      ✓ sell - fail - price unsafe (2592ms)
      ✓ sell - fail - price limit (2210ms)
      ✓ sell - success (2447ms)
      ✓ buy and sell - success (5193ms)
      ✓ sell - fail - deadline (1510ms)
      ✓ addLiquidity - fail - no marginBalance (2259ms)
      ✓ addLiquidity - fail - unsafe (2327ms)
      ✓ addLiquidity - success (1701ms)
      ✓ removeLiquidity - fail - shareBalance limited (535ms)
      ✓ removeLiquidity - success (3500ms)
      ✓ removeLiquidity - no position on removing liqudity (5353ms)
      ✓ removeLiquidity - transfer share (3848ms)
```

```
      ✓ updateIndex (989ms)
    settle
0.000000000000923114
      ✓ settle (1018ms)
    case review
      ✓ sellAndWithdraw0408 (3404ms)

  Contract: TestBrokerage
    ✓ exceptions (92ms)
    ✓ new user => broker works immediately (107ms)
    ✓ modify broker => new broker works later (205ms)
    ✓ when modifing => duplicated modify broker (the same broker) => delay timer is
reset (310ms)
    ✓ when modifing => duplicated modify broker (another broker) => delay timer is
reset (293ms)
    ✓ modify broker success => set the same broker => ignore (387ms)
    ✓ modify broker success => set another broker => new broker works later (370ms)

  Contract: TestCollateral
    exceptions
      ✓ constructor - invalid decimals (74ms)
      ✓ constructor - decimals out of range (63ms)
      ✓ withdraw - negtive amount (313ms)
      ✓ withdraw - negtive amount (352ms)
    misc
      ✓ getCashBalance (253ms)
      ✓ depositToProtocol (366ms)
    deposit / withdraw ether
      ✓ deposit (247ms)
      ✓ withdraw (317ms)
    deposit / withdraw
      ✓ invalid decimals (238ms)
      ✓ decimals == 18 (501ms)
      ✓ decimals == 8 (493ms)
      ✓ decimals == 5 (499ms)
    deposit / withdraw
      deposit
        ✓ deposit (235ms)
        ✓ deposit too much (80ms)
      withdraw
        ✓ withdraw with no application (88ms)
        ✓ withdraw with application but too early (127ms)
        ✓ withdraw with application but still too early (181ms)
        ✓ withdraw (251ms)
      exit
        ✓ exit repeat (183ms)
        ✓ exit (140ms)
        ✓ exit all (302ms)
    cash flow
      updateBalance
        ✓ updateBalance (232ms)
        ✓ updateBalance (344ms)
```

```
    transferBalance
      ✓ normal (93ms)
      ✓ too much (95ms)
      ✓ transfer 0 (77ms)

Contract: contractReader
  ✓ getGovParams (156ms)
  ✓ getPerpetualStorage (179ms)
  ✓ getAccountStorage (277ms)

Contract: exchange-amm
  exceptions
    ✓ taker order is maker only (225ms)
    ✓ invalid trading lot size (305ms)
    ✓ taker overfilled (408ms)
  trades
    ✓ buy (3023ms)
    ✓ buy - success - pnl < 0, critical deposit amount (1975ms)
    ✓ buy - fail - pnl < 0, lower than critical deposit amount (3337ms)
    ✓ sell - success (2947ms)
    ✓ buy and sell - success (6139ms)

Contract: exchange-user
  ✓ trade (752ms)
  ✓ soft fee (3860ms)
  ✓ soft fee - hit floor (6254ms)
  ✓ trade 1v1 (2677ms)
  ✓ close (4458ms)
  ✓ fail to match (970ms)
  ✓ maker only (606ms)
  ✓ invalid broker (730ms)
  ✓ invalid broker (726ms)
  ✓ invalid signature (996ms)
  exceptions
    ✓ wrong status (844ms)
    ✓ self trade (810ms)
    ✓ invalid side (826ms)
    ✓ market order cannot be maker (826ms)
    ✓ taker overfilled (1017ms)
    ✓ maker overfilled (1195ms)
    ✓ invalid trading lot size (1055ms)
    ✓ maker margin (2730ms)
    ✓ cancel order (64ms)
    ✓ taker margin (3027ms)
    ✓ dev safe (1963ms)
    ✓ no dev (1546ms)
    ✓ dev unsafe (2195ms)
    ✓ maker unsafe (4863ms)
    ✓ taker unsafe (5156ms)
    ✓ market order (2145ms)
  trades
    ✓ validate (2482ms)
```

```
      ✓ trade 1v1, trading size (3319ms)
      ✓ trade 1v1 (2608ms)
      ✓ broker balance (2636ms)
      ✓ broker unsafe (3559ms)
      ✓ dev unsafe (2329ms)
      ✓ validate (1948ms)
      ✓ cancel order (77ms)

  Contract: TestPerpGovernance
    exceptions
      ✓ amm required
      ✓ setGovernanceParameter exceptions (818ms)
      ✓ setGovernanceAddress exceptions (100ms)
    global config
      ✓ set governance value (124ms)
      ✓ key not exists (43ms)
    set parameters
      ✓ set dev address (69ms)
      ✓ set global config (161ms)
      ✓ set funding (108ms)
      ✓ set governance value (665ms)
      ✓ key not exists (58ms)
      ✓ not owner (43ms)
    status
      ✓ set governance value (611ms)
      ✓ key not exists (55ms)
    status
      ✓ beginGlobalSettlement (111ms)
      ✓ beginGlobalSettlement again (210ms)
      ✓ not owner (109ms)

  Contract: amm
    create amm
      ✓ should success (2252ms)
    trading
      ✓ addLiquidity - no position on removing liqudity (1966ms)
      ✓ removeLiquidity - no position on removing liqudity (5165ms)
      ✓ buy - success (2833ms)
      ✓ sell - success (2905ms)

  Contract: TestExtension
    trade
      ✓ buy (2318ms)
      ✓ sell (2404ms)

  Contract: testMath
    ✓ exceptions (307ms)
        R: 1851851851851851851666
        S: 1851851851851851853333
DIFF RANGE: 3333
    ✓ frac1 (82ms)
    ✓ frac2 neg (80ms)
```

```
    ✓ frac3 neg (86ms)
    ✓ roundHalfUp (158ms)
    ✓ unsigned wmul - trivial (149ms)
    ✓ unsigned wmul - overflow
    ✓ unsigned wmul - rounding (91ms)
    ✓ unsigned wdiv - trivial (78ms)
    ✓ unsigned wdiv - div by 0
    ✓ unsigned wdiv - rounding (134ms)
    ✓ signed wmul - trivial (486ms)
    ✓ signed wmul - overflow (49ms)
    ✓ signed wmul - rounding (423ms)
    ✓ signed wdiv - trivial (380ms)
    ✓ signed wdiv - div by 0
    ✓ signed wdiv - rounding (683ms)
    ✓ power (1291ms)
    ✓ log (911ms)
    ✓ logBase (585ms)
    ✓ ceil (122ms)
    ✓ max

  Contract: order
    ✓ test order (361ms)
    ✓ test order 2 (378ms)
    ✓ test order 3 (397ms)
    ✓ test order 3 (48ms)

  Contract: TestPerpetual
    tradePosition
      ✓ tradePosition - settlement (306ms)
      ✓ invalid side (59ms)
      ✓ transferCashBalance exceptions (93ms)
    liquidate
      ✓ partial liquidate - lot size (298ms)
      ✓ partial liquidate - nothing to liquidate (909ms)
      ✓ partial liquidate - add social loss (8163ms)
      ✓ liquidate - long pos (1083ms)
      ✓ liquidate - short pos (1101ms)
      ✓ liquidate 4 (1349ms)
    division
      ✓ :( (1653ms)
    collateral - ether
      ✓ insurance fund (263ms)
      ✓ withdrawEther initial (392ms)
      ✓ withdrawEther whitelist (949ms)
      ✓ fallback (169ms)
      ✓ depositEther (673ms)
      ✓ depositEtherAndSetBroker - 0 (125ms)
      ✓ depositEtherAndSetBroker (150ms)
    collateral - erc20
      ✓ insurance fund (561ms)
      ✓ withdraw initial (476ms)
      ✓ withdraw - whitelist (1094ms)
```

```
      ✓ deposit (223ms)
      ✓ depositFor (335ms)
      ✓ accounts (238ms)
      ✓ deposit && broker - 0 (131ms)
      ✓ deposit && broker (415ms)
      ✓ withdraw - with no application (481ms)
      ✓ withdraw - deposit + withdraw (609ms)
      ✓ withdraw - pnl = positive, withdraw until IM (2261ms)
    miscs
      ✓ transfer balance (573ms)
      ✓ settle (526ms)
    settlement
      ✓ settle (168ms)
      ✓ settle at wrong stage (63ms)
      ✓ settle at wrong stage 2 (177ms)
    trade
      ✓ fill margin up to im (3289ms)
      ✓ fill margin up to im (2831ms)
      ✓ withdraw (3015ms)
      ✓ buy (4088ms)
      ✓ buy (2859ms)
      ✓ sell (4115ms)
      ✓ isIMSafe (1314ms)

  Contract: exchange-user-reverse
    ✓ validate (831ms)
    ✓ trade 1v1 (2739ms)
    case review
      ✓ inversePosition0409 (4318ms)

  Contract: amm
    ✓ buy - success (4070ms)
    ✓ privileges (596ms)

  Contract: TestPosition
    exceptions
      ✓ addSocialLossPerContractPublic (42ms)
      ✓ trade (199ms)
    miscs
      ✓ get position (302ms)
      ✓ position balance (666ms)
      ✓ funding loss long (356ms)
      ✓ funding loss short (346ms)
      ✓ total size (453ms)
      ✓ socialloss - long (433ms)
      ✓ socialloss - short (440ms)
      ✓ remargin - 0 (57ms)
      ✓ remargin - long (676ms)
      ✓ remargin - short (710ms)
    liquidate
      ✓ without loss - long (836ms)
      ✓ without loss - short (780ms)
```

```
      ✓ with loss and funding - long (965ms)
      ✓ with loss and funding - short (961ms)
      ✓ handleSocialLoss (289ms)
      ✓ handleSocialLoss 2 (269ms)
      ✓ calculateLiquidateAmount long (458ms)
      ✓ liquidate (1213ms)
      ✓ liquidate more - long (1161ms)
      ✓ liquidate more - short (1174ms)
      ✓ liquidate more 2 - long (1201ms)
      ✓ liquidate more 2 - short (1135ms)
    social loss
      ✓ set long loss (124ms)
      ✓ add short loss (141ms)
    position size, margin
      ✓ basic info (2604ms)
      ✓ buy (712ms)
      ✓ sell (706ms)
      ✓ buy 1 + sell 1 (804ms)
      ✓ buy 1 + sell 0.5 + sell 0.5 (1183ms)
      ✓ buy 1 + buy 1.5 + sell 3.5 (1071ms)
    pnl
      ✓ without loss - 0 (234ms)
      ✓ without loss - long (814ms)
      ✓ with loss - long (1907ms)
      ✓ with loss - short (1885ms)
      ✓ with loss and funding - long (1867ms)
      ✓ with loss and funding - short (1958ms)
      ✓ buy 1 + sell 0.5 (1567ms)
    withdraw
      ✓ applyForWithdrawalPublic (2014ms)

  Contract: one block
    AMM: one block transactions
      ✓ index updated between 2 trades (4927ms)
      ✓ index updated before liquidate (4375ms)

  Contract: signature
    ✓ should be an valid signature (EthSign) (56ms)
    ✓ should be an valid signature (EIP712) (46ms)
    ✓ should be an invalid signature (EthSign) (43ms)
    ✓ should be an invalid signature (EIP712) (42ms)
    ✓ should revert when using an invalid signature type
    ✓ isValidSignature (38ms)
    ✓ isValidSignature 712 (52ms)
    ✓ isValidSignature invalid (44ms)
1589366656
    ✓ expire at (38ms)
    ✓ generate signature (176ms)
    ✓ generate invalid signature (186ms)

  Contract: statement
    ✓ setCashBalance (1761ms)
```

```
    ✓ set socialloss on settling (2855ms)
    ✓ set balance on settling (2489ms)
    ✓ set balance on settling 2 (2652ms)
    ✓ set balance on settling 2 (2711ms)
    ✓ settling forbids (3493ms)
    ✓ settling allows (1965ms)
    ✓ settled forbids (3663ms)
    ✓ settled allows (2587ms)
    ✓ settling liquidate (2666ms)
    ✓ settled liquidate (2097ms)


  317 passing (16m)

--------------------------|----------|----------|----------|----------|--------------
--|
File                      |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered
Lines |
--------------------------|----------|----------|----------|----------|--------------
--|
 exchange/                |      100 |    98.57 |      100 |      100 |
|
  Exchange.sol            |      100 |    98.57 |      100 |      100 |
|
 global/                  |      100 |      100 |      100 |      100 |
|
  GlobalConfig.sol        |      100 |      100 |      100 |      100 |
|
 lib/                     |      100 |      100 |      100 |      100 |
|
  LibEIP712.sol           |      100 |      100 |      100 |      100 |
|
  LibMath.sol             |      100 |      100 |      100 |      100 |
|
  LibOrder.sol            |      100 |      100 |      100 |      100 |
|
  LibSignature.sol        |      100 |      100 |      100 |      100 |
|
  LibTypes.sol            |      100 |      100 |      100 |      100 |
|
 liquidity/               |      100 |    94.38 |      100 |      100 |
|
  AMM.sol                 |      100 |     93.9 |      100 |      100 |
|
  AMMGovernance.sol       |      100 |      100 |      100 |      100 |
|
 perpetual/               |    99.47 |     94.3 |      100 |    99.46 |
|
  Brokerage.sol           |      100 |      100 |      100 |      100 |
|
  Collateral.sol          |      100 |     92.5 |      100 |      100 |
|
```

```
  Perpetual.sol           |      100 |     92.5 |      100 |      100 |
|
  PerpetualGovernance.sol |       98 |    98.28 |      100 |     97.5 |
75 |
  Position.sol            |    99.23 |     92.5 |      100 |    99.22 |
63 |
 proxy/                   |      100 |       75 |      100 |      100 |
|
  PerpetualProxy.sol      |      100 |       75 |      100 |      100 |
|
 reader/                  |      100 |      100 |      100 |      100 |
|
  ContractReader.sol      |      100 |      100 |      100 |      100 |
|
 token/                   |      100 |      100 |      100 |      100 |
|
  ShareToken.sol          |      100 |      100 |      100 |      100 |
|
--------------------------|----------|----------|----------|----------|--------------
--|
All files                 |    99.82 |    95.44 |      100 |    99.81 |
|
--------------------------|----------|----------|----------|----------|--------------
--|
```

Appendix 3 - Disclosure

ConsenSys Diligence (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD 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 ConsenSys and CD 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 Reports. ConsenSys and CD assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.