HMXv2 Review

Security Review

Cantina Managed review by:
Riley Holterhus, Lead Security Researcher
Throttle, Security Researcher
0x4non, Associate Security Researcher

September 27, 2023

Contents

1 Introduction **3**
1.1 About Cantina . . . 3
1.2 Disclaimer . . . 3
1.3 Risk assessment . . . 3
1.3.1 Severity Classification . . . 3

2 Security Review Summary **4**

3 Findings **5**
3.1 Critical Risk . . . 5
3.1.1 The `minProfitDuration` mechanism might harm honest users . . . 5
3.2 High Risk . . . 5
3.2.1 Updating a `MarketConfig` can break reserve accounting . . . 5
3.2.2 Inaccurate calculation of `_unrealizedPnlE30` for multiple positions . . . 6
3.2.3 Permanent reward accrual after liquidation . . . 7
3.3 Medium Risk . . . 7
3.3.1 Deleverage will revert for markets in closed hours . . . 7
3.3.2 Execution fees are not managed correctly by `Ext01Handler` . . . 7
3.3.3 Order queues might become stuck . . . 8
3.3.4 Position's unrealized funding fee is miscalculated . . . 9
3.4 Low Risk . . . 9
3.4.1 The `closeDelistedMarketPositions()` function invokes `nonReentrant` guard twice . . . 9
3.4.2 AUM calculation is missing `hlpLiquidityDebtUSDE30` . . . 9
3.4.3 Funding values not updated before calling `withdrawFundingFeeSurplus()` . . . 10
3.4.4 Wrong fee rate in `_decreasePosition()` . . . 10
3.4.5 Incorrect array length validation . . . 10
3.5 Gas Optimization . . . 11
3.5.1 Consider using Solady's `max`, `min`, `abs` functions in `HMXLib` . . . 11
3.5.2 Unnecessary `min()` function when decreasing position . . . 11
3.5.3 Unnecessary `else if` condition with `!vars.positionIsLong` . . . 12
3.5.4 Unnecessary `vars.order.reduceOnly` condition . . . 12
3.5.5 `_getHLPValueE30()` can be optimized by using named return and avoiding initialization variables . . . 13
3.5.6 Use `immutable` on variables that won't be changed . . . 13
3.5.7 Remove redundant variable definition `user` in `TraderLoyaltyCredit` . . . 14
3.5.8 Unreachable condition in `HMXLib.getSubAccount()` . . . 14
3.6 Informational . . . 14
3.6.1 Consider using named mappings to improve code readability . . . 14
3.6.2 `IHLP` and `ITraderLoyaltyCredit` interfaces should inherit `IERC20` . . . 15
3.6.3 Can add additional `nonReentrant` guards . . . 15
3.6.4 Missing attribution to original source . . . 16
3.6.5 `validateDeleverage()` might revert with underflow instead of `ITradeService_HlpHealthy()` . . . 16
3.6.6 Interfaces missing certain functions . . . 17
3.6.7 Interfaces missing `view` on certain functions . . . 18
3.6.8 Duplicate assignment in `TradeHelper.sol` . . . 19
3.6.9 Incorrect data type for `decimals` in `GetCollateralValue` struct . . . 19
3.6.10 Slippage protection in `SwitchCollateralRouter.execute()` . . . 19
3.6.11 The `isMax` oracle boolean is not used nor fully correct . . . 19
3.6.12 Carefully consider important config changes . . . 20
3.6.13 `nonReentrant` modifier considerations . . . 20
3.6.14 Fees are accrued when the market is closed . . . 20
3.6.15 Config param `maxFundingRate` name is misleading . . . 21
3.6.16 Funding rate changes during closed market . . . 21
3.6.17 Wrong naming with `closeLong` and `closeShort` booleans . . . 21
3.6.18 MEV considerations . . . 22
3.6.19 Remove unused events and errors . . . 22
3.6.20 Remove unused libraries and imports . . . 22
3.6.21 Use defined interfaces instead of importing contracts . . . 23

3.6.22 Use EIP-165 standard interface detection for contract interface checks 23

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Directly* exploitable security vulnerabilities that need to be fixed.
High	Security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All high issues should be addressed.
Medium	Objective in nature but are not security vulnerabilities. Should be addressed unless there is a clear reason not to.
Low	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.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. When determining the severity one first needs to determine whether the finding is subjective or objective. All subjective findings are considered of Minor severity.

Next it is determined whether the finding can be regarded as a security vulnerability. Some findings might be objective improvements that need to be fixed, but do not impact the project's security overall (Medium).

Finally, objective findings of security vulnerabilities are classified as either critical or major. Critical findings should be directly vulnerable and have a high likelihood of being exploited. Major findings on the other hand may require specific conditions that need to be met before the vulnerability becomes exploitable.

2 Security Review Summary

HMXv2 is an innovative pool-based perpetual DEX protocol designed to offer a range of advanced features. It introduces multi-asset collateral support and cross-margin flexibility, providing traders with enhanced options and opportunities.

The protocol incorporates secured measurements, including virtual price impact and funding fees, to ensure the protection of liquidity providers (LPs) from being overly exposed to a single direction. By implementing these measures, HMXv2 aims to create a more resilient and balanced trading environment.

From Aug. 14th - Sept. 8th the Cantina team conducted a review of HMXv2 on commit hash e7bdee...693f2b. The team identified a total of **43** issues in the following risk categories:

- Critical Risk: 1
- High Risk: 3
- Medium Risk: 4
- Low Risk: 5
- Gas Optimizations: 8
- Informational: 22

3 Findings

3.1 Critical Risk

3.1.1 The `minProfitDuration` mechanism might harm honest users

Severity: Critical Risk

Context: Calculator.sol#L1103-L1142

Description: To prevent exploiting sudden price changes (which are potentially due to manipulation), HMX has implemented a `minProfitDuration` mechanism. Essentially, the protocol will temporarily zero out any profit from positions that have been increased in the last `minProfitDuration` seconds.

Although this mechanism will prevent profits from short-term exploits, it could also affect honest users who are unaware of its consequences. Some problems might include:

1. Losing unrealized profit. If an already profitable position is increased, the existing profit will be zero for the next `minProfitDuration` seconds. If the user is not aware of this and calls `decreasePosition()` within this timeframe, they will have accidentally burned their initial profit.
2. Being closer to liquidation. Even if the user doesn't call `decreasePosition()` in the above example, their position is temporarily worth less. This means the user is closer to liquidation, which they might not be aware of.
3. Being unable to call `increasePosition()` at all. With the above two scenarios, it's possible the initial `increasePosition()` transaction doesn't even succeed because the unrealized profit is necessary for the initial margin requirement. This might not be the best UX, and the user may not expect a revert.

Recommendation: Ideally, a position's *existing* profit could be maintained after an increase. However, this might be difficult to implement correctly, especially considering the total pnl calculation which uses average entry prices.

Instead, the HMX team has decided to remove the `minProfitDuration` mechanism in certain situations. In particular, when a position's equity is calculated (i.e. in `increasePosition()` or `liquidate()`), the profit is never zeroed out. Additionally, it is now impossible to increase or decrease a position that is profitable and has been increased in the past `minProfitDuration` seconds.

HMX: As explained, fixed in PR 322.

Cantina: Verified.

3.2 High Risk

3.2.1 Updating a `MarketConfig` can break reserve accounting

Severity: High Risk

Context: TradeService.sol#L848-L880

Description: In HMX, each position has a reserved maximum profit. For the purposes of tracking HLP utilization and borrowing fees, the sums of these reserved amounts are stored in `GlobalState.reserveValueE30` and `AssetClass.reserveValueE30`. Naturally, when a position's size is decreased/increased, these sums must be updated as well.

Within `_decreasePosition()`, this is implemented roughly as follows:

```
_vars.globalState.reserveValueE30 -=
  (_vars.position.reserveValueE30 * _vars.positionSizeE30ToDecrease) /
  _vars.absPositionSizeE30;
_vars.assetClass.reserveValueE30 -=
  (_vars.position.reserveValueE30 * _vars.positionSizeE30ToDecrease) /
  _vars.absPositionSizeE30;

// ... omitting code here ...

_vars.position.reserveValueE30 =
          ((_temp.newAbsPositionSizeE30 * _marketConfig.initialMarginFractionBPS *
↪  _marketConfig.maxProfitRateBPS) /
            BPS) /
          BPS;
```

With this code, the cumulative values are decreased proportional to the existing position reserve. On the other hand, the position's new reserve is completely recalculated based on its new size. So, if the market's `initialMarginFractionBPS` or `maxProfitRateBPS` values have changed since the position was last updated, the cumulative reserve amounts and the position's reserve amount will change by different values.

In the worst case, the cumulative sums can become smaller than the sum of all position reserves, preventing some positions from being closed/liquidated (due to arithmetic underflow).

Recommendation: Ensure that the cumulative amounts stay in sync with actual position reserve amounts. To do this, consider an initial subtraction of the entire old position reserve, followed by an addition of the entire new position reserve. This is similar to how the `AccumSE` and `AccumS2E` values are tracked.

HMX: Fixed in PR 314.

Cantina: Verified.

3.2.2 Inaccurate calculation of `_unrealizedPnlE30` for multiple positions

Severity: High Risk

Context: Calculator.sol#L649-L656

Description: The `_getUnrealizedPnlAndFee()` function applies the `pnlFactorBps` (currently 80%) to each profitable position, while unprofitable positions have their loss fully deducted. So, for example, a user with a $12k unrealized gain position and a $10k unrealized loss position would have its unrealized pnl calculated as `12_000 * 0.8 - 10_000 == -400`. This is incorrect, because the user's equity is being reduced, even though they have an overall gain.

Recommendation: Apply the `pnlFactorBps` to the subaccount's *overall* pnl if it is positive:

```
  if (_var.isProfit) {
    if (_var.delta >= _var.position.reserveValueE30) {
        _var.delta = _var.position.reserveValueE30;
    }
-   _unrealizedPnlE30 += int256((pnlFactorBps * _var.delta) / BPS);
+   _unrealizedPnlE30 += int256( _var.delta);
  } else {
    _unrealizedPnlE30 -= int256(_var.delta);
  }

  // ... omitting code here ...

+   if( _unrealizedPnlE30 > 0) {
+     _unrealizedPnlE30 = (pnlFactorBps * _unrealizedPnlE30 / BPS);
+   }

    return (_unrealizedPnlE30, _unrealizedFeeE30);
  }
```

With the previous example, this would give the user an extra equity of `0.8 * (12_000 - 10_000) == +1_600`.

HMX: Fixed in PR 304.

Cantina: Verified.

3.2.3 Permanent reward accrual after liquidation

Severity: High Risk

Context: LiquidationService.sol

Description: When a subaccount is liquidated, each position is wound down completely. In normal circumstances, a position being closed would trigger the protocol's `onDecreasePosition()` hooks, but this is missing during liquidation.

Currently, these hooks manage rewards accounting. In particular, `TradingStakingHook.onDecreasePosition()` is responsible for removing the position from the staking pool. So, if a position is liquidated, the user can permanently accumulate and harvest rewards.

Recommendation: Trigger the `onDecreasePosition()` hooks when a subaccount is liquidated.

HMX: Fixed in PR 304.

Cantina: Verified.

3.3 Medium Risk

3.3.1 Deleverage will revert for markets in closed hours

Severity: Medium Risk

Context: TradeService.sol#L626

Description: The deleveraging safety mechanism is designed to automatically close positions in a specific order, prioritising the most profitable positions first. However, if a position is opened in a market that is currently closed, the deleveraging flow will be reverted.

Recommendation: Allow deleveraging positions in markets that are in closed hours.

HMX: Fixed in commit 477dd83.

Cantina: Verified.

3.3.2 Execution fees are not managed correctly by `Ext01Handler`

Severity: Medium Risk

Context: Ext01Handler.sol#L150-L184

Description: In the `Ext01Handler` contract, users are intended to compensate the keeper with a `msg.value` that is exactly equal to `minExecutionOrderOf[_params.orderType]`. This approach is slightly different from the other handlers, where `msg.value` must equal a user-supplied `_params.executionFee` value (which must be larger than a certain threshold).

Moreover, the `Ext01Handler` currently hardcodes the final order's `executionFee` as 0. This is likely a mistake and would result in the keeper receiving no compensation while the contract's ETH remains unused.

Recommendation: Consider matching the behavior of the other handlers by making the following changes:

```
  uint128 _minExecutionFee = minExecutionOrderOf[_params.orderType];
  if (_params.executionFee < _minExecutionFee) revert IExt01Handler_InsufficientExecutionFee();
- if (msg.value != _minExecutionFee) revert IExt01Handler_InCorrectValueTransfer();
+ if (msg.value != _params.executionFee) revert IExt01Handler_InCorrectValueTransfer();

  // ... omitting code here ...

  orders.push(
    GenericOrder({
      orderId: uint248(_orderId),
      status: OrderStatus.PENDING,
      createdTimestamp: uint48(block.timestamp),
      executedTimestamp: 0,
      orderType: uint24(_params.orderType),
-     executionFee: uint128(0),
+     executionFee: _params.executionFee,
      rawOrder: _rawOrder
    })
  );
```

HMX: Fixed in PR 326.

Cantina: Verified.

3.3.3 Order queues might become stuck

Severity: Medium Risk

Context: CrossMarginHandler.sol, Ext01Handler.sol, LiquidityHandler.sol

Description: In the `CrossMarginHandler`, `Ext01Handler` and `LiquidityHandler` contracts, there exists a `nextExecutionOrderIndex` variable. These variables point to the next order that needs to be processed, so these contracts essentially maintain queues of operations that the keeper must follow. To prevent these queues from becoming stuck, each order is processed in a try-catch block, so reverts can be caught and skipped past.

However, there still exist two ways the queue can theoretically become stuck:

1. If the code outside the try block reverts. For example, with `LiquidityHandler`, the contract refunds the user's deposit if an error is caught. If the deposit was in USDC, it is technically possible that the user has since been added to the token's blacklist, in which case the transfer will fail and the whole queue becomes stuck.
2. If someone intentionally wastes gas for the entire transaction. Most of the external calls do not specify an explicit `gas` amount, so a malicious contract will have access to most of the remaining gas amount. A malicious contract could potentially waste this gas and/or revert with a large error message. In both scenarios, the calling contract can be forced to revert with the out-of-gas error, which would also freeze the queue. Note that this is not currently a problem, since no malicious contracts can gain control flow in this way.

Recommendation: Consider addressing each of these individual issues, or alternatively remove the queues altogether.

To address the first issue, consider removing the actual ERC20 transfer, and instead credit the user a balance they can withdraw later. To address the second issue, consider setting explicit `gas` amounts on external calls, and use a library like ExcessivelySafeCall to carefully handle large revert messages.

To remove the queue altogether, consider changing these handlers to be similar to `LimitTradeHandler`.

HMX: Fixed in PR 326 and PR 327.

Cantina: Verified. The HMX team has decided to remove the queues altogether. For the `CrossMarginHandler` and `LiquidityHandler` specifically, new V2 contracts have been created. Since these new contracts have a different storage layout, they are planned to be deployed behind entirely new proxies.

3.3.4 Position's unrealized funding fee is miscalculated

Severity: Medium Risk

Context: Calculator.sol#L673-687

Description: To keep track of funding fee payments, a checkpoint value is stored in each position (`Position.lastFundingAccrued`) and each market (`Market.fundingAccrued`). Within the `_getUnrealizedPnlAndFee()` function, the code intends to calculate the *position's* unrealized funding fee, but it does so using the *market's* checkpoint value:

```
int256 lastFundingAccrued = _market.fundingAccrued;
// ... omitting code here ...
_unrealizedFeeE30 += getFundingFee(_var.position.positionSizeE30, currentFundingAccrued, lastFundingAccrued);
```

This will lead to a miscalculation, especially if the funding rate has significantly changed since the position was last updated. Since `_getUnrealizedPnlAndFee()` is only used within the `getEquity()` calculation, this will only affect decisions based on a position's health, and can't actually be abused for a profit.

Recommendation: Use the position's checkpoint value instead of the market's checkpoint value:

```
- int256 lastFundingAccrued = _market.fundingAccrued;
+ int256 lastFundingAccrued = _vars.position.lastFundingAccrued;
```

HMX: Fixed in PR 304.

Cantina: Verified.

3.4 Low Risk

3.4.1 The `closeDelistedMarketPositions()` function invokes `nonReentrant` guard twice

Severity: Low Risk

Context: BotHandler.sol#L333-L376

Description: In the `BotHandler` contract, the `closeDelistedMarketPositions()` function can close many delisted market positions in one transaction (versus `closeDelistedMarketPosition()` which can only close one). However, it is actually impossible to call `closeDelistedMarketPositions()`, since the function internally uses `_closeDelistedMarketPositions()`, and both functions have `nonReentrant` modifiers.

Note that the HMX team independently discovered this issue during the review period.

Recommendation: Remove the `nonReentrant` modifier from the internal `_closeDelistedMarketPositions()` function.

HMX: As noted, this has been fixed in commit 77f327b.

Cantina: Verified.

3.4.2 AUM calculation is missing `hlpLiquidityDebtUSDE30`

Severity: Low Risk

Context: Calculator.sol#L74-L95

Description: The `getAUME30()` function calculates the total HLP value, including HLP liquidity, unrealized counter-trading pnl, and unrealized borrowing fees.

It would also make sense to include `hlpLiquidityDebtUSDE30` in this calculation. This is the amount of funding fees paid from the HLP when the accumulated trading funding fees are insufficient. Subsequent funding fee payments will always reduce this debt, and the entire amount belongs to the HLP, so it would be appropriate to add it to the AUM calculation.

Recommendation: Add `_vaultStorage.hlpLiquidityDebtUSDE30()` to the sum in `getAUME30()`.

HMX: Fixed in PR 319.

Cantina: Verified.

3.4.3 Funding values not updated before calling `withdrawFundingFeeSurplus()`

Severity: Low Risk

Context: CrossMarginService.sol#L232-L247

Description: In `withdrawFundingFeeSurplus()`, a calculation of the surplus funding fee is done, which depends on each market's `accumFundingLong` and `accumFundingShort` values. These values change every second (based on funding rate velocity), but the current implementation doesn't ensure that up-to-date values are being used. As a result, the calculation might not be perfectly accurate, and might take too much or too little funding.

Recommendation: For better accounting, consider initially calling `tradeHelper.updateFundingRate(i)` for each market index `i`.

HMX: Fixed in commit b45f8ba and commit b0f2a63.

Cantina: Verified.

3.4.4 Wrong fee rate in `_decreasePosition()`

Severity: Low Risk

Context: TradeService.sol#L773-L782

Description: There are separate fee rates for increasing positions (`increasePositionFeeRateBPS`) and decreasing positions (`decreasePositionFeeRateBPS`). The `_decreasePosition()` function is currently using the rate for increasing positions, which is incorrect.

Recommendation: In `_decreasePosition()`, use the `_marketConfig.decreasePositionFeeRateBPS` value instead of the `_marketConfig.increasePositionFeeRateBPS` value.

HMX: Fixed in commit 4211639.

Cantina: Verified.

3.4.5 Incorrect array length validation

Severity: Low Risk

Context: LimitTradeHandler.sol#L574-575, ConfigStorage.sol#L332-L334

Description: In two separate locations, the protocol intends to enforce that three different arrays all have the same length. However, the implementation is incorrectly using `&&` in place of `||`, and thus might not revert if one length is different than the other two.

Recommendation: Change the `&&` to `||`. For example:

```
- if (_accounts.length != _subAccountIds.length && _accounts.length != _orderIndexes.length)
+ if (_accounts.length != _subAccountIds.length || _accounts.length != _orderIndexes.length)
      revert ILimitTradeHandler_InvalidArraySize();
```

HMX: Fixed in commit 456ea98 and commit 230f2f2.

Cantina: Verified.

3.5 Gas Optimization

3.5.1 Consider using Solady's `max`, `min`, `abs` functions in `HMXLib`

Severity: Gas Optimization

Context: HMXLib.sol#L15-L29

Description: The `HMXLib` contains functions `max`, `min`, and `abs`. The current implementations of these functions might not be the most gas-efficient. There's a potential opportunity to improve gas efficiency by using Solady's implementations.

Recommendation: Consider using the Solady library for the implementations of `max`, `min`, and `abs` functions.

HMX: Fixed in commit 13d3170.

Cantina: Verified.

3.5.2 Unnecessary `min()` function when decreasing position

Severity: Gas Optimization

Context: LimitTradeHandler.sol#L783-L786, LimitTradeHandler.sol#L831-L834

Description: In the `LimitTradeHandler`, there is a special case when a position is being decreased, but not decreased enough to flip the position to the opposite side. In this special case, the `_positionSizeE30ToDecrease` variable is being set to the minimum of `vars.sizeDelta` and `_existingPosition.positionSizeE30` (adjusting for negatives appropriately). This is technically not necessary, because in this case the `sizeDelta` can't be greater than `positionSizeE30` in absolute terms.

Recommendation: Remove the unnecessary min logic as follows:

```
  if (vars.order.reduceOnly) {
    // ... omitting code here ...
  } else {
    if (vars.sizeDelta > 0) {
      if (vars.isNewPosition || vars.positionIsLong) {
        // ... omitting code here ...
      } else if (!vars.positionIsLong) {
        bool _flipSide = !vars.order.reduceOnly && vars.sizeDelta > (-_existingPosition.positionSizeE30);
        if (_flipSide) {
          // ... omitting code here ...
        } else {
          // Not flip
          _tradeService.decreasePosition({
            _account: vars.order.account,
            _subAccountId: vars.order.subAccountId,
            _marketIndex: vars.order.marketIndex,
-           _positionSizeE30ToDecrease: HMXLib.min(
-             uint256(vars.sizeDelta),
-             uint256(-_existingPosition.positionSizeE30)
-           ),
+           _positionSizeE30ToDecrease: uint256(vars.sizeDelta),
            _tpToken: vars.order.tpToken,
            _limitPriceE30: _isGuaranteeLimitPrice ? vars.order.triggerPrice : 0
          });
        }
      }
    } else if (vars.sizeDelta < 0) {
      // ... omitting code here ...
      } else if (vars.positionIsLong) {
        bool _flipSide = !vars.order.reduceOnly && (-vars.sizeDelta) > _existingPosition.positionSizeE30;
        if (_flipSide) {
          // ... omitting code here ...
        } else {
          // Not flip
          _tradeService.decreasePosition({
            _account: vars.order.account,
            _subAccountId: vars.order.subAccountId,
            _marketIndex: vars.order.marketIndex,
-           _positionSizeE30ToDecrease: HMXLib.min(
-             uint256(-vars.sizeDelta),
-             uint256(_existingPosition.positionSizeE30)
```

```
-         ),
+         _positionSizeE30ToDecrease: uint256(-vars.sizeDelta),
            _tpToken: vars.order.tpToken,
            _limitPriceE30: _isGuaranteeLimitPrice ? vars.order.triggerPrice : 0
          });
        }
      }
    }
  }
```

Note that this only works because the `!vars.order.reduceOnly` condition is redundant, which is discussed in the issue titled "Unnecessary `vars.order.reduceOnly` condition".

HMX: Fixed in commit bc2c683.

Cantina: Verified.

3.5.3 Unnecessary `else if` condition with `!vars.positionIsLong`

Severity: Gas Optimization

Context: LimitTradeHandler.sol#L756

Description: In the `LimitTradeHandler`, there is a large nested `if` statement, with one section roughly as follows:

```
if (vars.isNewPosition || vars.positionIsLong) {
    // ... omitting code here ...
} else if (!vars.positionIsLong) {
    // ... omitting code here ...
}
```

In the `else if` branch, it is already known that `vars.positionIsLong` is false, since the prior condition was false.

Recommendation: Since the `else if` will always be true once reached, replace the `else if` with simply `else`.

HMX: Fixed in commit d297488.

Cantina: Verified.

3.5.4 Unnecessary `vars.order.reduceOnly` condition

Severity: Gas Optimization

Context: LimitTradeHandler.sol#L757, LimitTradeHandler.sol#L805

Description: In the `LimitTradeHandler`, there is a large nested `if` statement, with the first statement branching on the boolean `vars.order.reduceOnly`. Within two sub-branches of the `else` case, there are these two lines:

```
bool _flipSide = !vars.order.reduceOnly && vars.sizeDelta > (-_existingPosition.positionSizeE30);
```

```
bool _flipSide = !vars.order.reduceOnly && (-vars.sizeDelta) > _existingPosition.positionSizeE30;
```

Since `vars.order.reduceOnly` is already known to be `false` here, the `!vars.order.reduceOnly` condition is unnecessary.

Recommendation: Remove the unnecessary condition.

HMX: Fixed in commit 3deb59b.

Cantina: Verified.

3.5.5 `_getHLPValueE30()` can be optimized by using named return and avoiding initialization variables

Severity: Gas Optimization

Context: Calculator.sol#L147-L163

Description: By utilizing the named return feature, it is possible to avoid initializing values with a default value (`0`). This, in combination with eliminating the use of the `value` variable and encompassing the operations within the `unchecked` block, will save on gas and enhance code readability.

Recommendation:

```
  contract Calculator is OwnableUpgradeable, ICalculator {

    /// @notice GetHLPValue in E30
    /// @param _isMaxPrice Use Max or Min Price
-   /// @return HLP Value
-   function _getHLPValueE30(bool _isMaxPrice) internal view returns (uint256) {
+   /// @return assetValue HLP Value
+   function _getHLPValueE30(bool _isMaxPrice) internal view returns (uint256 assetValue) {
      ConfigStorage _configStorage = ConfigStorage(configStorage);

      bytes32[] memory _hlpAssetIds = _configStorage.getHlpAssetIds();
-     uint256 assetValue = 0;
      uint256 _len = _hlpAssetIds.length;

-     for (uint256 i = 0; i < _len; ) {
-       uint256 value = _getHLPUnderlyingAssetValueE30(_hlpAssetIds[i], _configStorage, _isMaxPrice);
-       unchecked {
-         assetValue += value;
-         ++i;
+     unchecked {
+       for (uint256 i; i < _len; ++i) {
+       assetValue += _getHLPUnderlyingAssetValueE30(_hlpAssetIds[i], _configStorage, _isMaxPrice);
      }
    }

-   return assetValue;
  }
```

HMX: Fixed in commit 824bfa04.

Cantina: Verified.

3.5.6 Use `immutable` on variables that won't be changed

Severity: Gas Optimization

Context: OrderReader.sol#L14-L17

Description: The `OrderReader` contract declares the following state variables: `configStorage`, `limitTradeHandler`, `oracleMiddleware` and `perpStorage`. As there is no function provided to modify these variables post-deployment, it would be more gas-efficient to mark them as immutable.

Recommendation: Mark `configStorage`, `limitTradeHandler`, `oracleMiddleware`, and `perpStorage` as immutable.

HMX: Fixed in commit f85c48e.

Cantina: Verified.

3.5.7 Remove redundant variable definition `user` in `TraderLoyaltyCredit`

Severity: Gas Optimization

Context: TraderLoyaltyCredit.sol#L112-L113, TraderLoyaltyCredit.sol#L135-L136, TraderLoyaltyCredit.sol#L157-L158, TraderLoyaltyCredit.sol#L176-L177, TraderLoyaltyCredit.sol#L196-L197

Description: Throughout several functions in `TraderLoyaltyCredit`, an `address` variable called `user` is assigned, but only used in a single place. Consider simplifying this, since the variable definition is not necessarily needed.

Recommendation: Remove `address user = msg.sender;` and use `msg.sender`

HMX: Fixed in commit 6554623.

Cantina: Verified.

3.5.8 Unreachable condition in `HMXLib.getSubAccount()`

Severity: Gas Optimization

Context: HMXLib.sol#L10-L11

Description: The condition `if (_subAccountId > 255)` within the `getSubAccount()` function is always false because the variable `_subAccountId` is of type `uint8` (which can never exceed the value 255).

Recommendation: Remove this condition check and the associated error `HMXLib_WrongSubAccountId()`, as it's superfluous and can never be triggered.

```
  library HMXLib {
-   error HMXLib_WrongSubAccountId();
    function getSubAccount(address _primary, uint8 _subAccountId) internal pure returns (address _subAccount) {
-     if (_subAccountId > 255) revert HMXLib_WrongSubAccountId();
      return address(uint160(_primary) ^ uint160(_subAccountId));
  }
```

HMX: Fixed in commit 8b98d43 and commit 996bdc2.

Cantina: Verified.

3.6 Informational

3.6.1 Consider using named mappings to improve code readability

Severity: Informational

Context: HLP.sol#L15 and TLCStaking.sol#L28-L30

Description: There are instances where mappings are declared without named key types. While this is syntactically correct, using named key types might enhance the readability of the code and make it easier for other developers to understand the purpose and type of the keys being mapped.

Recommendation: It's recommended to use named key types for the mappings for better code clarity. For example, instead of `mapping(address => bool) public minters;`, consider using `mapping(address minterAddress => bool isMinter) public minters;`. Implementing this change will potentially improve code readability without affecting the contract's logic.

HMX: Fixed in commit 863876e.

Cantina: Verified.

3.6.2 `IHLP` and `ITraderLoyaltyCredit` interfaces should inherit `IERC20`

Severity: Informational

Context: IHLP.sol#L4, ITraderLoyaltyCredit.sol#L4

Description: Both the `HLP` and `TraderLoyaltyCredit` contracts are tokens `ERC20`. However, their respective interfaces are not fully defined as `ERC20` nor do they expose all the `ERC20` methods. It's recommended that these interfaces directly inherit the `IERC20` interface to ensure all functionalities of `ERC20` are available and exposed.

Recommendation: Update both `IHLP` and `ITraderLoyaltyCredit` interfaces to inherit from the IERC20 interface, thereby ensuring the availability of all ERC20 functionalities.

HMX: Fixed in commit 6554623.

Cantina: Verified.

3.6.3 Can add additional `nonReentrant` guards

Severity: Informational

Context: General

Description: In some places, a contract has `nonReentrant` modifiers on some functions but not others, even if the functions are all similar in nature. It might make sense to add `nonReentrant` modifiers to the following functions:

- `Ext01Handler`:
  - `setCrossMarginService()`
  - `setLiquidationService()`
  - `setLiquidityService()`
  - `setTradeService()`
  - `setPyth()`
  - `setOrderExecutor()`
- `LimitTradeHandler`:
  - `setDelegate()`
  - `setGuaranteeLimitPrice()`
  - `setTradeService()`
  - `setMinExecutionTimestamp()`
  - `setLimitTradeHelper()`
- `LiquidityHandler`:
  - `setHlpStaking()`
- `RebalanceHLPHandler`:
  - `addGlp()`
  - `setWhiteListExecutor()`
- `LiquidationService`:
  - `liquidate()`
- `PerpStorage`:
  - `removePositionFromSubAccount()`
  - `updateGlobalLongMarketById()`
  - `updateGlobalShortMarketById()`
  - `updateGlobalState()`

    - `updateAssetClass()`
    - `updateMarket()`
    - `decreaseReserved()`
    - `increasePositionSize()`
    - `decreasePositionSize()`
- `VaultStorage`:
    - most functions

In addition to this, the following contracts do not inherit the reentrancy guard contract, even though similar contracts might have:

- `LimitTradeHelper`
- `RebalanceHLPService`
- `ConfigStorage`
- All staking contracts
- All strategy contracts
- All oracle contracts

Recommendation: Consider adding additional `nonReentrant` guards to some (or all) of the above. This would be more for consistency, as these modifiers are not strictly needed (as described in the issue titled "`nonReentrant` modifier considerations").

HMX: We'll leave it as is.

Cantina: Acknowledged.

3.6.4 Missing attribution to original source

Severity: Informational

Context: TickMath.sol#L1-L6, FullMath.sol#L1-L12

Description: Certain portions of the code appear to be directly taken from external libraries: Uniswap's v3-core TickMath.sol and from Uniswap's v3-core FullMath.sol

Recommendation: Add a comment or note in the code mentioning the original source from Uniswap's repository for clarity and transparency.

HMX: Fixed in commit 7dd7058 and commit 10f9be0

Cantina: Verified.

3.6.5 `validateDeleverage()` might revert with underflow instead of `ITradeService_HlpHealthy()`

Severity: Informational

Context: TradeService.sol#L669-L671

Description: The `validateDeleverage()` function is intended to revert if deleveraging is not necessary (because HLP is already healthy). This is implemented as follows:

```
uint256 _aum = _calculator.getAUME30(false);
uint256 _tvl = _calculator.getHLPValueE30(false);

if ((_tvl - _aum) * BPS <= (BPS - ConfigStorage(configStorage).getLiquidityConfig().hlpSafetyBufferBPS) * _tvl)
  revert ITradeService_HlpHealthy();
```

Technically, `_tvl - _aum` may be negative, for example, if the HLP has a large unrealized profit. In this case, the code will correctly revert, but with an underflow error and not the intended `ITradeService_HlpHealthy()` error.

Recommendation: Consider adding a specific check for `_tvl < _aum`, so that the more specific error will be used:

```
if (
  (_tvl < _aum) ||
  ((_tvl - _aum) * BPS <= (BPS - ConfigStorage(configStorage).getLiquidityConfig().hlpSafetyBufferBPS) * _tvl)
)
  revert ITradeService_HlpHealthy();
```

HMX: Fixed in commit 317dbbe.

Cantina: Verified.

3.6.6 Interfaces missing certain functions

Severity: Informational

Context: IVaultStorage.sol, IConfigStorage.sol, ITradeService.sol, ILiquidityService.sol, ILiquidationService.sol, ICrossMarginService.sol, ITradeHelper.sol, ICalculator.sol

Description: Some interfaces are missing function declarations for functions/variables that exist in the actual implementation. The list of missing function declarations is:

- IVaultStorage:
  - lossDebt()
  - tradingFeeDebt()
  - borrowingFeeDebt()
  - fundingFeeDebt()
  - subTradingFeeDebt()
  - subBorrowingFeeDebt()
  - subFundingFeeDebt()
  - subLossDebt()
  - convertFundingFeeReserveWithHLP()
  - withdrawSurplusFromFundingFeeReserveToHLP()
- IConfigStorage:
  - minimumPositionSize()
  - getAssetClassConfigsLength()
- ITradeService:
  - calculator()
- ILiquidityService:
  - addLiquidity()
  - validatePreAddRemoveLiquidity()
- ILiquidationService:
  - vaultStorage()
- ICrossMarginService
  - switchCollateral()
- ITradeHelper:
  - perpStorage()
  - vaultStorage()
  - configStorage()
  - increaseCollateral()
  - decreaseCollateral()

- updateFeeStates()

- ICalculator:
  - getNextBorrowingRate()
  - getFundingFee()
  - getBorrowingFee()

Recommendation: Add the missing function declarations to these interfaces.

HMX: Fixed IVaultStorage in commit 82637f6. Fixed IConfigStorage in commit 33ce015. Fixed ITradeService in commit 4d1c7a6. Fixed ILiquidityService and another part of ITradeService in commit 04c990c. Fixed ILiquidationService in commit da44bfd. Fixed ICrossMarginService in commit e93c238. Fixed ITradeHelper in commit c49871f. Fixed ICalculator in commit 78c1d24.

Cantina: Verified.

3.6.7 Interfaces missing view on certain functions

Severity: Informational

Context: IVaultStorage.sol, ILiquidityService.sol, ICrossMarginService.sol, ICalculator.sol

Description: In some interfaces, there are functions declared without the view keyword, even if the actual function/variable implementation would imply it is view. The list of functions with this problem is:

- IVaultStorage:
  - globalBorrowingFeeDebt()
  - globalLossDebt()
- ILiquidityService:
  - configStorage()
  - vaultStorage()
  - perpStorage()
- ICrossMarginService:
  - calculator()
  - configStorage()
  - vaultStorage()
- ICalculator:
  - oracle()
  - vaultStorage()
  - configStorage()
  - perpStorage()
  - getAUME30()
  - getHLPPrice()
  - getAddLiquidityFeeBPS()
  - getRemoveLiquidityFeeBPS()
  - getSettlementFeeRate()

Recommendation: Add the view keyword to these functions.

HMX: Fixed IVaultStorage in commit 82637f6. Fixed ILiquidityService and ICalculator in commit 04c990c. Fixed ICrossMarginService in commit e93c238.

Cantina: Verified.

3.6.8 Duplicate assignment in `TradeHelper.sol`

Severity: Informational

Context: TradeHelper.sol#L473-L474

Description: In `TradeHelper._increaseCollateral()`, the `_vars.vaultStorage` variable is assigned twice in a row, both times with the exact same value (`VaultStorage(vaultStorage)`). This seems to be a minor oversight.

Recommendation: Remove the duplicate assignment.

HMX: Fixed in commit b189c37

Cantina: Verified.

3.6.9 Incorrect data type for `decimals` in `GetCollateralValue` struct

Severity: Informational

Context: Calculator.sol#L557

Description: The `GetCollateralValue` struct defines `decimals` as a `uint256`. However, according to EIP20-decimals, the correct data type for decimals should be `uint8`.

Recommendation: Change the data type of decimals from `uint256` to `uint8` to align with EIP-20 standards.

HMX: Fixed in commit 958d9b2.

Cantina: Verified.

3.6.10 Slippage protection in `SwitchCollateralRouter.execute()`

Severity: Informational

Context: CrossMarginService.sol#L323, SwitchCollateralRouter.sol#L27-L46

Description: The `SwitchCollateralRouter` and associated dexters don't enforce slippage protection themselves. Instead, the slippage protection happens within the `switchCollateral()` function in `CrossMarginService`. This serves as the primary mechanism to prevent unfavorable exchanges due to price fluctuations (MEV arbitrage/sandwich attacks).

Recommendation: Although the protocol's current control flow will always have slippage protection, keep in mind that the router and dexters themselves are not safe. Developers and integrators should be aware of this behavior and not use these contracts directly.

HMX: Noted.

Cantina: Acknowledged.

3.6.11 The `isMax` oracle boolean is not used nor fully correct

Severity: Informational

Context: LiquidityService.sol#L317-L320, PythAdapter.sol#L77, StakedGlpOracleAdapter.sol#L38

Description: When the protocol requests an oracle price, the call to the `OracleMiddleware` specifies an `isMax` boolean. This is supposed to represent whether the price should be maximized or minimized, presumably accounting for price spreads and integer rounding.

However, this boolean is not actually used in the downstream components of the oracle. Moreover, some of the oracle calls seem to use an incorrect value, such as `LiquidityService._exitPool()` requesting a min price, which would benefit the user over the protocol.

Recommendation: It appears this is a legacy feature that is no longer relevant. Consider removing the boolean in future updates.

HMX: Noted. We will remove this in the future update.

Cantina: Acknowledged.

3.6.12 Carefully consider important config changes

Severity: Informational

Context: ConfigStorage.sol

Description: Within the `ConfigStorage` contract, there are admin functions that can update very important protocol parameters. Changing a few of these parameters can significantly affect the entire protocol.

For instance, changing the HLP token address (through `setHLP()`) would change the token that is minted/burned by the protocol, affecting all existing LPs. Another important parameter is the `pnlFactorBPS` (updated through `setPnlFactor()`), and a decrease in this value could instantly liquidate existing positions. There are several other parameters that are similarly powerful.

Recommendation: Exercise caution when changing important config parameters, and consider the downstream consequences of each change.

HMX: Noted.

Cantina: Acknowledged.

3.6.13 `nonReentrant` modifier considerations

Severity: Informational

Context: General

Description: The `nonReentrant` modifier is currently being used throughout the codebase. Technically, these modifiers are not strictly necessary, as during execution there are no external calls that are untrusted (even the DEX router paths and ERC20 tokens must be whitelisted). Also, if the user *could* gain control flow, they wouldn't be able to re-enter the protocol, since most execution is managed by the keeper address.

On the other hand, if the `nonReentrant` modifiers become more important in a future update, it should be noted that they don't protect against *cross-contract* reentrancy. This type of reentrancy would be difficult to completely mitigate, because some degree of cross-contract communication is necessary.

Recommendation: Since the `nonReentrant` modifiers exist as a defensive measure, it makes sense to keep them in the codebase, even if they aren't strictly necessary. In future updates to the protocol, consider if cross-contract reentrancy problems might be introduced.

HMX: Noted. We do not think that this would pose serious threat and it would require a lot of changes. So, we'll leave it as is.

Cantina: Acknowledged.

3.6.14 Fees are accrued when the market is closed

Severity: Informational

Context: TradeHelper.sol#L440-L444

Description: The funding and borrowing fees are still accrued when markets are closed (which is possible for equities, FX, and commodities markets). This is an intentional design choice and is justified because the HLP is still exposed to after-hours events, even if the price impact of these events can't be realized until later.

Recommendation: No recommendation. This issue serves a public information role.

HMX: We decided to accrue funding fee even during the closed hours due to there is still a risk to HLP/the system while the market is closed i.e. political shifted, wars, etc. which make prices fluctuate even when markets are closed. This is for both protecting HLP and also disincentivized traders to leave positions open during closed hours.

Cantina: Acknowledged.

3.6.15 Config param `maxFundingRate` name is misleading

Severity: Informational

Context: IConfigStorage.sol#L47

Description: The name `maxFundingRate` is misleading. Actually, funding rates are unbounded. A more meaningful name would be `maxFundingRateChange` or `maxFundingRateUpdate`.

Recommendation: Rename the struct field to a more meaningful name.

HMX: Fixed in commit 1805562.

Cantina: Verified.

3.6.16 Funding rate changes during closed market

Severity: Informational

Context: TradeHelper.sol#L263-L265

Description: The funding rate calculation and funding rate velocity mechanism can suffer (or be manipulated) because it takes into account the time when markets are closed (which is possible for equities, FX, and commodities markets).

When the market resumes on Monday after Friday close, `_calculator.proportionalElapsedInDay()` will return `2.5 * e18` (i.e. 2.5 days). If the market ends up with a large skew during the weekend, it can considerably change the funding rate. Such a large change can be difficult to cool off, as it would take exactly 2.5 days of the opposite skew to cancel out.

Recommendation: Consider implementing a system where closed trading hours do not increase or decrease funding rates.

HMX: My thought is we should let it run. If we keep funding rate at the same rate during market close time, this will happen:

- Traders make market skew to Long and drive funding rate high up.
- Just before the market close, exploiter will open large short position. It can be so large that it shift market skew the other way around.
- That large short position will keep earning funding fee with the same rate during the market close time without risk of liquidation.

If we keep this feature as is and let the funding rate velocity run, the funding rate will gradually shift to the unfavorable side of the exploiter and it would protect against this kind of attack.

Cantina: In summary, since funding fees do accumulate during closed hours, it makes sense that the rates change during this period too. If not, the system might be exploitable as HMX has described. See the issue titled "Fees are accrued when the market is closed" for further discussion on why it is appropriate to accumulate funding fees when the market is closed.

3.6.17 Wrong naming with `closeLong` and `closeShort` booleans

Severity: Informational

Context: LimitTradeHandler.sol#L725-L726

Description: In the `LimitTradeHandler`, the `closeLong` boolean is true if and only if the order is decreasing a *short* position, and the `closeShort` boolean is true if and only if the order is decreasing a *long* position. The naming seems to be the opposite of what was intended.

Recommendation: Switch the names of the variables.

HMX: Fixed in commit 80b664a.

Cantina: Verified.

3.6.18 MEV considerations

Severity: Informational

Context: Calculator.sol#L74-L95, Calculator.sol#L1144-L1160

Description: The HMX protocol has been designed in a way that mitigates most MEV/frontrunning issues. However, it is difficult to reduce these issues completely, and there are two considerations potentially worth highlighting:

1. The HLP receives its deposit/withdrawal fees, surplus funding fees, and trading fees all in discrete batches. It might be possible to extract value by minting HLP immediately before these transactions and burning HLP immediately after. There may be similar discrete jumps with the `cook()` strategies or in HLP rebalances.
2. The overall unrealized pnl calculation is not perfect.

   Firstly, the calculation doesn't consider if a position goes underwater, goes past its maximum profit, or has its profit limited by its `lastIncreaseTimestamp`. Since these concepts *do* affect the pnl that each individual position contributes, there can be discrete jumps in value when certain positions are interacted with.

   Secondly, the calculation doesn't consider that the pnl of one position will change once another position closes/opens (due to adaptive pricing). If a few large positions are closed/opened in a short timeframe, a jump in value can occur, even if there isn't any actual pnl.

In most scenarios, these inefficiencies are small relative to the entire protocol. So, any potential profit would likely be outweighed by the associated protocol/transaction fees.

Recommendation: Keep this behavior in mind, and potentially reassess if these inefficiencies become profitable to exploit in the future.

HMX: Noted by HMX team.

Cantina: Acknowledged.

3.6.19 Remove unused events and errors

Severity: Informational

Context: TraderLoyaltyCredit.sol#L27-L28, TLCStaking.sol#L24-L25, TLCStaking.sol#L20

Description: `TraderLoyaltyCredit` declares events `FeedReward` and `Claim`, but these events are not being used. Similarly, the `TLCStaking` contract declares errors `TLCStaking_BadDecimals`, `TLCStaking_DuplicateStakingToken`, and `TLCStaking_UnknownStakingToken`, but they are not being used.

Recommendation: Remove the unused events `FeedReward` and `Claim` from `TraderLoyaltyCredit`. Remove the unused errors `TLCStaking_BadDecimals`, `TLCStaking_DuplicateStakingToken`, and `TLCStaking_UnknownStakingToken` from `TLCStaking`.

HMX: Fixed in commit 6554623 and commit 49d662.

Cantina: Verified.

3.6.20 Remove unused libraries and imports

Severity: Informational

Context: TraderLoyaltyCredit.sol#L7, ConfigStorage.sol#L10, TraderLoyaltyCredit.sol#L13, Ext01Handler.sol#L23, ICalculator.sol#L5, LiquidityHandler.sol#L16-L20, CrossMarginHandler.sol#L19, TLCStaking.sol#L11

Description: The OpenZeppelin library, `SafeERC20Upgradeable`, is imported both in `TraderLoyaltyCredit.sol` and `ConfigStorage.sol` but is not used. Similarly, the `VaultStorage` contract is imported into `Ext01Handler.sol`, `ICalculator.sol` and `CrossMarginHandler.sol` without being used. In `LiquidityHandler.sol`, the contracts `VaultStorage`, `PerpStorage`, `Calculator`, `OracleMiddleware`, and `HLP` are imported but remain unused.

Recommendation: Remove the unused library and contracts.

HMX: Fixed in commit 6554623, commit 766b263, commit 8ec6ad9.

Cantina: Verified.

3.6.21 Use defined interfaces instead of importing contracts

Severity: Informational

Context: Calculator.sol#L12-L15, BotHandler.sol#L24-L30, CrossMarginHandler.sol#L19-L20, Ext01Handler.sol#L17-L24, LimitTradeHandler.sol#L16-L19, LiquidityHandler.sol#L14-L20, TradeHelper.sol#L16, CrossMarginService.sol#L18-L21, LiquidationService.sol#L13-L17, LiquidityService.sol#L13-L18, TradeService.sol#L14-L19

Description: Across various files in the project, there are instances where the actual contracts are imported instead of their respective interfaces.

Recommendation: When one contract makes an external call to another, consider using the target contract's interface instead of importing the entire contract directly.

HMX: Noted. Importing raw contracts is our code convention.

Cantina: Acknowledged. The HMX team has chosen to keep their current convention.

3.6.22 Use EIP-165 standard interface detection for contract interface checks

Severity: Informational

Context: Calculator.sol#L60, BotHandler.sol, CrossMarginHandler.sol, LimitTradeHandler.sol, Ext01Handler.sol, LiquidityHandler.sol, TradeHelper.sol, StakedGlpOracleAdapter.sol#L28, LeanPyth.sol, TradeService.sol, CrossMarginService.sol, LiquidationService.sol, LiquidityService.sol#L97, EpochFeedableRewarder.sol#L66, TLCStaking.sol#L49, FeedableRewarder.sol#L65, TLCHook.sol#L43, TradingStakingHook.sol#L29, ConfigStorage.sol#L256

Description: There are many sanity checks present during the `initialize()` and `update()` functions, to confirm the expected contract interface at a given address. While this approach provides a level of assurance, EIP-165 is a standardized method for contracts to declare which interfaces they support, and this might be a more efficient and standardized approach.

Recommendation: Implement the EIP-165 Standard Interface Detection to replace the current sanity checks. This standard provides a robust way for contracts to declare which interfaces they support and for others to query this support. Link to EIP-165 for further reference.

HMX: Noted. We will implement this in our new smart contracts. We are going to leave the already the deployed ones as they are.

Cantina: Acknowledged.