hexens × QUICKSWAP

Apr.23

SMART CONTRACT AUDIT REPORT FOR QUICKSWAP

CONTENTS

CONTENTS

ABOUT HEXENS

Hexens is a cybersecurity company that strives to elevate the standards of security in Web 3.0, create a safer environment for users, and ensure mass Web 3.0 adoption.

Hexens has multiple top-notch auditing teams specialized in different fields of information security, showing extreme performance in the most challenging and technically complex tasks, including but not limited to: Infrastructure Audits, Zero Knowledge Proofs / Novel Cryptography, DeFi and NFTs. Hexens not only uses widely known methodologies and flows, but focuses on discovering and introducing new ones on a day-to-day basis.

In 2022, our team announced the closure of a $4.2 million seed round led by IOSG Ventures, the leading Web 3.0 venture capital. Other investors include Delta Blockchain Fund, Chapter One, Hash Capital, ImToken Ventures, Tenzor Capital, and angels from Polygon and other blockchain projects.

Since Hexens was founded in 2021, it has had an impressive track record and recognition in the industry: Mudit Gupta - CISO of Polygon Technology - the biggest EVM Ecosystem, joined the company advisory board after completing just a single cooperation iteration. Polygon Technology, 1inch, Lido, Hats Finance, Quickswap, Layerswap, 4K, RociFi, as well as dozens of DeFi protocols and bridges, have already become our customers and taken proactive measures towards protecting their assets.

AUDIT LED BY

KASPER ZWIJSEN

Lead Smart Contract Auditor | Hexens

Audit Starting Date
16.03.2023

Audit Completion Date
04.04.2023

---

hexens × QUICKSWAP

METHODOLOGY

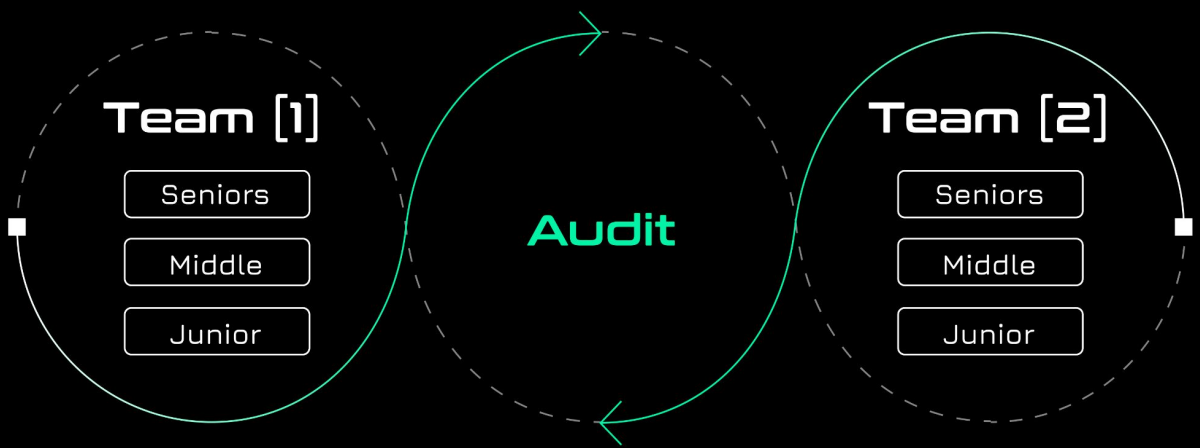
COMMON AUDIT PROCESS

Companies often assign just one engineer to one security assessment with no specified level. Despite the possible impeccable skills of the assigned engineer, it carries risks of the human factor that can affect the product's lifecycle.

Auditor* Audit

HEXENS METHODOLOGY

Hexens methodology involves 2 teams, including multiple auditors of different seniority, with at least 5 security engineers. This unique cross-checking mechanism helps us provide the best quality in the market.



SEVERITY STRUCTURE

The vulnerability severity is calculated based on two components

- Impact of the vulnerability
- Probability of the vulnerability

IMPACT	PROBABILITY			
	Rare	Unlikely	Likely	Very Likely
Low / Info	Low / Info	Low / Info	Medium	Medium
Medium	Low / Info	Medium	Medium	High
High	Medium	Medium	High	Critical
Critical	Medium	High	Critical	Critical

SEVERITY CHARACTERISTICS

Vulnerabilities can range in severity and impact, and it's important to understand their level of severity in order to prioritize their resolution. Here are the different types of severity levels of vulnerabilities:

CRITICAL

Vulnerabilities with this level of severity can result in significant financial losses or reputational damage. They often allow an attacker to gain complete control of a contract, directly steal or freeze funds from the contract or users, or permanently block the functionality of a protocol. Examples include infinite mints and governance manipulation.

HIGH

Vulnerabilities with this level of severity can result in some financial losses or reputational damage. They often allow an attacker to directly steal yield from the contract or users, or temporarily freeze funds. Examples include inadequate access control integer overflow/underflow, or logic bugs.

MEDIUM

Vulnerabilities with this level of severity can result in some damage to the protocol or users, without profit for the attacker. They often allow an attacker to exploit a contract to cause harm, but the impact may be limited, such as temporarily blocking the functionality of the protocol. Examples include uninitialized storage pointers and failure to check external calls.

LOW

Vulnerabilities with this level of severity may not result in financial losses or significant harm. They may, however, impact the usability or reliability of a contract. Examples include slippage and front-running, or minor logic bugs.

INFORMATIONAL

Vulnerabilities with this level of severity are regarding gas optimizations and code style. They often involve issues with documentation, incorrect usage of EIP standards, best practices for saving gas, or the overall design of a contract. Examples include not conforming to ERC20, or disagreement between documentation and code.

It's important to consider all types of vulnerabilities, including informational ones, when assessing the security of the project. A comprehensive security audit should consider all types of vulnerabilities to ensure the highest level of security and reliability.

EXECUTIVE SUMMARY

OVERVIEW

This audit covered QuickSwap's new perpetual futures exchange called QuickPerps. The protocol is a fork of GMX and both adds and removes some functionality, such as the integration of a new pricing oracle that makes use of the Pyth Network.

Our security assessment was a full review of the QuickPerps protocol and its smart contracts. We have thoroughly reviewed each contract individually, as well as the system as a whole.

During our audit, we have identified 1 critical severity vulnerability in the OrderBook. It would allow an attacker to drain all WETH, effectively stealing it from the orders of other users.

We have also identified 2 high severity vulnerabilities, various minor vulnerabilities and code optimisations.

Finally, all of our reported issues were fixed by the development team and consequently validated by us.

We can confidently say that the overall security and code quality have increased after completion of our audit.

SCOPE

The analyzed resources are located on:
https://github.com/QuickSwap/perps/commit/fdc0fb9d91761952c5ea3e2a3e5cde903f4483cc
https://github.com/QuickSwap/perps/commit/a8c0bebf43350ee4783df4f92bdcb0aa23d8c214

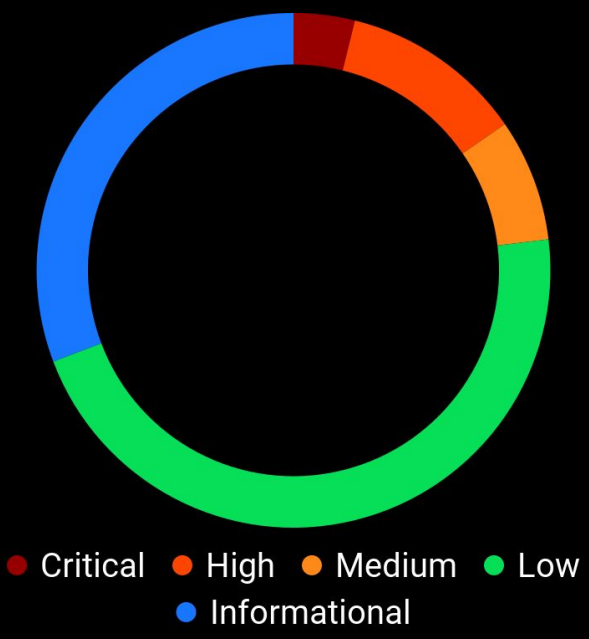
The issues described in this report were fixed in the following commits:
https://github.com/QuickSwap/perps/commit/547dd0e5af9317df68cbd459ae36405561d80e8b
https://github.com/QuickSwap/perps/commit/e534930c19e59575cf21402b2ecd50fb9f9e0195

SUMMARY

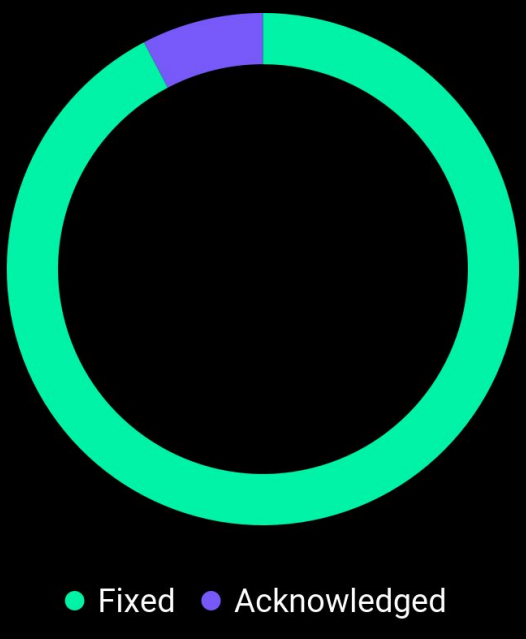
SEVERITY	NUMBER OF FINDINGS
CRITICAL	1
HIGH	3
MEDIUM	2
LOW	12
INFORMATIONAL	8

TOTAL: 26

SEVERITY



STATUS



WEAKNESSES

This section contains the list of discovered weaknesses.

QSWP-20. ORDERBOOK'S WETH CAN BE DRAINED

SEVERITY: Critical

PATH: core/OrderBook.sol:executeDecreaseOrder:L865-930

REMEDIATION: see description

STATUS: fixed

DESCRIPTION:

A decrease position order takes a **receiveToken** as parameter. Upon execution of the order, the function will try to swap the **amountOut** of the **collateralToken** for the **receiveToken**. This allows a user to withdraw other tokens instead of their initially deposited collateral token.

Another parameter is the **withdrawEth** parameter, which specifies whether the user wants the execution function to unwrap their WETH. In **createDecreaseOrder** it is checked whether the **receiveToken** or **collateralToken** is WETH if this boolean parameter is true.

In **executeDecreaseOrder** after the position has been decreased, the OrderBook would have **amountOut** in **collateralToken**. Then it checks whether a **receiveToken** was specified and if so, it checks the liquidity of the vault for **receiveToken** using **VaultUtils.getMaxAmountIn**. If there is not enough liquidity, it will skip the swap.

Then, if **withdrawEth** is true, it will unwrap WETH and transfer ETH to the user. If not, it will send the **receiveToken** if the swap succeeded and **collateralToken** otherwise.

However, this logic fails to consider the case where **withdrawEth** is true (and therefore the **receiveToken** would be WETH) and the swap is not performed due to insufficient liquidity.

In that case, the function would transfer out ETH with an amount equal to the **amountOut** for **collateralToken**. If **collateralToken** is a token with >= 18 decimals and a value lower than ETH (e.g. FRAX), then the user would receive a lot more value due to the sudden conversion to ETH (with the same amount).

Furthermore, the failing of the swap due to insufficient liquidity can be forced by an attacker in various ways. For example, the Vault contract allows for depositing and withdrawing of any asset. One can deposit token A and obtain QLP and immediately redeem the QLP for token B. By repeating this or using enough assets, they can decrease the pool amount for **receiveToken**. If the value of **amountOut collateralToken** is greater than this pool amount for **receiveToken**, then the swap would not be executed.

As a result, a malicious user is able to (repeatedly) steal the WETH belonging to other users from the OrderBook.

```
function executeDecreaseOrder(
    address _address,
    uint256 _orderIndex,
    address payable _feeReceiver
) external override nonReentrant {
    _onlyOrderExecutor();
    DecreaseOrder memory order = decreaseOrders[_address][_orderIndex];
    require(order.account != address(0), "OB: no order");

    // decrease long should use min price
    // decrease short should use max price
    (uint256 currentPrice, ) = validatePositionOrderPrice(order.triggerAboveThreshold, order.triggerPrice,
order.indexToken, !order.isLong, true);

    delete decreaseOrders[_address][_orderIndex];
    _removeFromOpenOrders(_address,_orderIndex,OrderType.DECREASE);

    uint256 amountOut = IRouter(router).pluginDecreasePosition(
        order.account,
        order.collateralToken,
        order.indexToken,
        order.collateralDelta,
        order.sizeDelta,
        order.isLong,
        address(this)
    );

    bool isSwapExecuted;

    if (order.receiveToken != address(0)) {
        IVaultUtils vaultUtils = IVault(vault).vaultUtils();

        uint256 maxIn =vaultUtils.getMaxAmountIn(order.collateralToken, order.receiveToken);

        if(amountOut<=maxIn){
            IERC20(order.collateralToken).safeTransfer(vault, amountOut);
            amountOut = _vaultSwap(order.collateralToken, order.receiveToken, order.minOut, address(this));
            isSwapExecuted = true;
        }
    }
```

```
if (order.withdrawETH) {
    _transferOutETHWithGasLimit(amountOut, payable(order.account));
  } else {
    if (order.receiveToken != address(0) && isSwapExecuted) {
      IERC20(order.receiveToken).safeTransfer(order.account, amountOut);
    }else{
      IERC20(order.collateralToken).safeTransfer(order.account, amountOut);
    }
  }

  _transferOutETH(order.executionFee, _feeReceiver);

  emit ExecuteDecreaseOrder(
    order.account,
    _orderIndex,
    order.collateralToken,
    order.collateralDelta,
    order.indexToken,
    order.sizeDelta,
    order.isLong,
    order.triggerPrice,
    order.triggerAboveThreshold,
    order.executionFee,
    currentPrice
  );
}
```

The edge case, where **collateralToken** is not WETH, **receiveToken** is WETH, **withdrawEth** is true and the swap was not executed, should be correctly handled.

For example:

```
if (order.withdrawETH && (isSwapExecuted || order.receiveToken == address(0))) {
  _transferOutETHWithGasLimit(amountOut, payable(order.account));
} else {
  if (order.receiveToken != address(0) && isSwapExecuted) {
    IERC20(order.receiveToken).safeTransfer(order.account, amountOut);
  } else {
    IERC20(order.collateralToken).safeTransfer(order.account, amountOut);
  }
}
```

QSWP-4. ORDER LIST ITERATION WILL QUICKLY BECOME IMPOSSIBLE DUE TO GAS USAGE

SEVERITY: High

PATH: libraries/utils/IterableMapping.sol

REMEDIATION: we would recommend to not keep track of which key was deleted through a deletion flag. Instead, the keys array should shrink whenever an order is deleted. This way iteration would only take actual active orders into account

STATUS: fixed

DESCRIPTION:

The IterableMapping library is used in **OrderBook.sol** to keep a list of all orders. This data structure also keeps a list of key structs, containing the actual order key and a deletion flag. Iteration happens over this list of keys.

Any time an order is created, the order is inserted into the mapping. When an order gets canceled or executed, the order is removed from the mapping.

However, the removal only removes the data and sets the deletion flag in the corresponding key index to **true**.

As a result, iteration will still always iterate over each order that has existed, even it has been deleted. This can be seen in **iterate_next** where it iterates over all keys and checks whether they were deleted.

Each deletion check is a storage read, which costs about 500 gas. Because this array never shrinks in size, the gas cost of this iteration will grow over time.

For example, even after only 1000 orders have been created and executed, the function will always consume at least 500k gas. After 60.000 orders, the gas cost would be 30m and it would not fit in one block, making it impossible to call.

The mapping will naturally grow during normal usage of the protocol, making the function more expensive over time. But an attacker could also force the growth of the mapping by creating and cancelling orders in a loop.

```
function remove(itmap storage self, uint256 key) internal returns (bool success) {
    uint256 keyIndex = self.data[key].keyIndex;
    if (keyIndex == 0)
        return false;
    delete self.data[key];
    self.keys[keyIndex - 1].deleted = true;
    self.size --;
}
function iterate_next(itmap storage self, uint256 keyIndex) internal view returns (uint256 r_keyIndex) {
    keyIndex++;
    while (keyIndex < self.keys.length && self.keys[keyIndex].deleted)
        keyIndex++;
    return keyIndex;
}
```

QSWP-19. EXECUTOR CAN BE GAS GRIEVED

SEVERITY: High

PATH: core/PositionRouter.sol:executeIncreasePositions (L224-257, L259-291)

REMEDIATION: check whether all tokens in the path parameter are whitelisted (through the Vault contract) for both increase and decrease position orders

STATUS: fixed

DESCRIPTION:

The function **executeIncreasePositions** is called automatically by the price oracles, such as **FastPriceFeed** and the newly added **PythPriceFeed**. It executes each increase position order using a try-catch statement and also cancels the order in a try-catch if it fails.

The **createIncreasePosition** function allows for creating an increase position order using a fake, malicious token in the first element of the **path** field. The executor will then execute this position order with **executeIncreasePosition** and inside of this it will call **transferFrom** on the malicious token as the first external call.

As a result, a malicious user can force the spending of all the gas of the executor's call when the **transferFrom** is called on the malicious token.

Transaction simulation (e.g. through **eth_estimateGas** and **eth_call**) does not protect from this type of attack. A contract is able to know whether execution is a simulation or a real transaction and act accordingly to hide malicious behaviour with regards to gas griefing.

```
function executeIncreasePositions(uint256 _endIndex, address payable _executionFeeReceiver) external
override onlyPositionKeeper {
  uint256 index = increasePositionRequestKeysStart;
  uint256 length = increasePositionRequestKeys.length;

  if (index >= length) { return; }

  if (_endIndex > length) {
    _endIndex = length;
  }

  while (index < _endIndex) {
    bytes32 key = increasePositionRequestKeys[index];

    try this.executeIncreasePosition(key, _executionFeeReceiver) returns (bool _wasExecuted) {
      if (!_wasExecuted) { break; }
    } catch {
      try this.cancelIncreasePosition(key, _executionFeeReceiver) returns (bool _wasCancelled) {
        if (!_wasCancelled) { break; }
      } catch {}
    }

    delete increasePositionRequestKeys[index];
    index++;
  }

  increasePositionRequestKeysStart = index;
}
```

QSWP-28. HIGH DELAYS IN API3 DAPI PRICE FEEDS

SEVERITY: High

PATH: core/VaultPriceFeed.sol

REMEDIATION: always make use of the secondary Pyth price feed and to monitor the used dAPI proxies for availability

STATUS: acknowledged, see commentary

DESCRIPTION:

The API3 dAPI proxies are used as price feeds in the VaultPriceFeed contract.

We found that the API3 dAPI proxies on zkEVM experience high delays and are infrequently updated. For example, at the moment the ETH/USD pair has not been updated for more than 10 hours: API3 Market

The prices of ETH and other crypto currencies are highly volatile, i.e. in 10 hours the price could vary a lot. This could potentially lead to exploitation due to malicious arbitrage.

```
function _getApi3Price(address _token) private view  returns (uint256) {
    address proxy = priceFeedProxies[_token];
    require(proxy != address(0), "VaultPriceFeed: invalid price feed proxy");
    (int224 price, uint256 timestamp) = IProxy(proxy).read();
    require(price > 0, "VaultPriceFeed: price not positive");
    require(
        timestamp + expireTimeForPriceFeed > block.timestamp,
        "VaultPriceFeed: expired"
    );
    return uint256(uint224(price));
}
```

Commentary from client:

“Integration issue with API3 was also discovered in the internal audit process. We work closely with the API3 engineering team to solve it without code changes. If API3 proxies should not be "fed" we had a 24h control built in which APi3 is completely disregarded until the controls are fulfilled again.”

QSWP-24. USERS' ETH COULD GET LOST

SEVERITY: Medium

PATH:
core/BasePositionManager.sol:_transferOutETHWithGasLimitIgnoreFail:L279-286

REMEDIATION: the send function to send ETH is deprecated and no longer recommend. We would recommend to use call with a gas limit instead, where the return value is checked to be true

STATUS: fixed

DESCRIPTION:

The function **_transferOutETHWithGasLimitIgnoreFail** is used in contracts that inherit from **BasePositionManager**, such as **PositionManager** and **PositionRouter**, to send ETH collateral back to the account or ETH fees to the fee receiver.

However, as the name suggests, it uses **address.send(value)** and the return value of whether the transfer was successful is not checked.

As a result, it becomes possible for users to lose their ETH assets if they use a smart contract without the receive or fallback function implemented.

```
function _transferOutETHWithGasLimitIgnoreFail(uint256 _amountOut, address payable _receiver) internal
{
  IWETH(weth).withdraw(_amountOut);

  // use `send` instead of `transfer` to not revert whole transaction in case ETH transfer was failed
  // it has limit of 2300 gas
  // this is to avoid front-running
  _receiver.send(_amountOut);
}
```

QSWP-27. LOW AVAILABILITY OF API3 PROXIES ON POLYGON ZKEVM

SEVERITY: Medium

PATH: core/VaultPriceFeed.sol

REMEDIATION: Quickswap would have to make sure that the dAPI proxies for the desired tokens are created and funded

STATUS: acknowledged, see commentary

DESCRIPTION:

Due to the complete unavailability of Chainlink on Polygon zkEVM, the VaultPriceFeed uses API3 as it's primary price oracle. However, we found that the total amount of available API3 dAPI proxies on Polygon zkEVM is also very low.

Currently, there are only 11 out of 118 funded dAPI proxies. This can be viewed on the API3 market with the 'Polygon zkEVM' and 'funded' filters: API3 Market

As a result, the Vault would not be able to support tokens that are missing proxies.

```
function _getApi3Price(address _token) private view  returns (uint256) {
    address proxy = priceFeedProxies[_token];
    require(proxy != address(0), "VaultPriceFeed: invalid price feed proxy");
    (int224 price, uint256 timestamp) = IProxy(proxy).read();
    require(price > 0, "VaultPriceFeed: price not positive");
    require(
        timestamp + expireTimeForPriceFeed > block.timestamp,
        "VaultPriceFeed: expired"
    );
    return uint256(uint224(price));
}
```

Commentary from client:

“Integration issue with API3 was also discovered in the internal audit process. We work closely with the API3 engineering team to solve it without code changes. If API3 proxies should not be "fed" we had a 24h control built in which APi3 is completely disregarded until the controls are fulfilled again.”

QSWP-25. CENTRALISATION RISK

SEVERITY: Low

PATH: see description

REMEDIATION: see description

STATUS: fixed

DESCRIPTION:

There are several functions for governance or admin accounts that have a lot of privileges and can directly or indirectly take assets from users.

The main point of control is the **Timelock** contract, where a single **admin** address is defined. If this **admin** address where to be an EOA and the private key got compromised, then they would be able:

1. Change the **gov** address of the **Vault** contract through **signalSetGov** and **setGov**. This would allow them to completely drain the vault and directly take user assets using **Vault.upgradeVault**.
2. Redeem USDQ for any asset from the vault using **redeemUsdq** and directly take user assets.
3. Change the price feed of the vault using **setPriceFeed** to manipulate the prices and indirectly take users assets.
4. Directly take user-staked tokens from the **RewardTracker** contract through **withdrawToken**.
5. Manipulate prices of assets in the vault through **setTokenConfig**.

6. Change the **gov** address of the **PositionRouter** and **PositionManager**, which would also allow to directly take user assets that were deposited using **approve** and **sendValue**.
7. Change the **gov** address of price feeds, which can lead to price manipulation and indirect theft of user assets.

We highly recommend to use a multi-sig as **admin** of the **Timelock** contract.

Furthermore, each **gov** address in the core contracts should only be a Timelock contract with a large enough period of pending time for proposals. This should give enough time to react to any exploitation.

If a **gov** address of some core contract is an EOA or multi-sig instead, than that would greatly increase the risk of direct exploitation if that address

```
contract Timelock is ITimelock {
  [...]
}
```

QSWP-26. PRICE ORACLE RISK

SEVERITY: Low

PATH: see description

REMEDIATION: see description

STATUS: fixed

DESCRIPTION:

Quickswap Perpetuals is a fork of GMX and works with the same concept of price feeds fed by oracle data. This introduces various risks concerning arbitrage (or MEV) or even price manipulation.

For example, in January 2023, the protocol Mycelium, which is also a fork of GMX, was exploited due to an issue with the price feed. Basically, their fast price feed consisted of 3 different oracles, of which one went offline and another one started reporting volatile prices for ETH-USD. As a result, it caused a discrepancy and someone exploited this as an arbitrage opportunity.

The Mycelium exploit was partly a result of data (or rather oracle) unavailability, but mostly due to inaccuracy (i.e. price volatility) and the small amount of oracles just exacerbated the volatile prices. From this we can conclude that the secondary oracle requires a good amount of oracles so that unavailability and volatility of some oracles do not affect the price too much.

The main risks are therefore concerning price data availability and accuracy, a price oracle might go offline or go stale, or it might report volatile prices.

Quickswap Perpetuals instead introduces Pyth price feeds instead of the fast price feeds. Pyth Network is itself an aggregator of price feed data and publishes this to an oracle contract on various chains. As such, this does solve the issue of price volatility.

However, we believe that this does fully solve the problem of data/oracle availability. Pyth prices can go stale and the price feed would fall back to the Chainlink price. Furthermore, Pyth oracle contracts are dependent on data from Pythnet through the Wormhole messaging bridge. Problems in the Wormhole bridge could potentially affect data availability of Pyth oracles.

Moreover, the specific chain that Quickswap Perpetuals should also be taken into account. In the case of Polygon zkEVM, we see that as of now Chainlink is not yet available, but Pyth is. However, these are new deployments and availability and accuracy should be monitored first.

We do believe that using both Chainlink and Pyth, as well as discarding stale prices, using spreads and falling back to Chainlink, will be more solid and should lead to higher price certainty.

```
contract PythPriceFeed is IPythPriceFeed, Governable {
  [...]
}
```

QSWP-2. ORDERBOOK AND POSITIONROUTER STRUCTS CAN BE PACKED

SEVERITY: Low

PATH: see description

REMEDIATION: move the bool elements to be adjacent and also line up with one address to be optimally packed

STATUS: fixed

DESCRIPTION:

We have identified the following structs that can be packed in order to save gas:

1. **core/OrderBook.sol:IncreaseOrder, DecreaseOrder, SwapOrder** (L33-68)
2. **core/PositionRouter.sol:IncreasePositionRequest, DecreasePositionRequest** (L17-48)

The structs are not optimally packed and it will require more storage writes to create/update a struct.

For example, each struct has 2-3 **bool** types, which can be combined into 1 storage slot if there are adjacent. Currently, each struct has a **bool** and **uint256** next to each other, causing each **bool** to take up an entire storage slot.

1:

```
struct IncreaseOrder {
    address account;
    address purchaseToken;
    uint256 purchaseTokenAmount;
    address collateralToken;
    address indexToken;
    uint256 sizeDelta;
    bool isLong;
    uint256 triggerPrice;
    bool triggerAboveThreshold;
    uint256 executionFee;
}
struct DecreaseOrder {
    address account;
    address collateralToken;
    address receiveToken;
    uint256 collateralDelta;
    address indexToken;
    uint256 sizeDelta;
    bool isLong;
    uint256 triggerPrice;
    bool triggerAboveThreshold;
    uint256 minOut;
    bool withdrawETH;
    uint256 executionFee;
}
struct SwapOrder {
    address account;
    address[] path;
    uint256 amountIn;
    uint256 minOut;
    uint256 triggerRatio;
    bool triggerAboveThreshold;
    bool shouldUnwrap;
    uint256 executionFee;
}
```

2:

```
struct IncreasePositionRequest {
    address account;
    address[] path;
    address indexToken;
    uint256 amountIn;
    uint256 minOut;
    uint256 sizeDelta;
    bool isLong;
    uint256 acceptablePrice;
    uint256 executionFee;
    uint256 blockNumber;
    uint256 blockTime;
    bool hasCollateralInETH;
    address callbackTarget;
}

struct DecreasePositionRequest {
    address account;
    address[] path;
    address indexToken;
    uint256 collateralDelta;
    uint256 sizeDelta;
    bool isLong;
    address receiver;
    uint256 acceptablePrice;
    uint256 minOut;
    uint256 executionFee;
    uint256 blockNumber;
    uint256 blockTime;
    bool withdrawETH;
    address callbackTarget;
}
```

QSWP-12. VARIABLES CAN BE MARKED AS IMMUTABLE

SEVERITY: Low

PATH: see description

REMEDIATION: see description

STATUS: fixed

DESCRIPTION:

We have identified the following locations where it would be beneficial to mark a variable as **immutable**:

1. **oracle/pyth/PythPriceFeed.sol:pyth** (L17): this variable is only set in the constructor and it can therefore be marked as immutable.

If a variable is **immutable** it will become a constant and part of the byte code upon creation of the contract. Any subsequent read will cost almost no gas, compared to an **sload** each time, saving a lot of gas.

```
IPyth public pyth;
```

QSWP-6. CONTRACT CAN BE ACCIDENTALLY OSSIFIED

SEVERITY: Low

PATH: OrderBook.sol:setGov:L318-322

REMEDIATION: add a check for _gov against address(0) and also to implement two-step ownership transferral, where the current owner sets a pending owner and the pending owner has to accept this before becoming the new owner

STATUS: fixed

DESCRIPTION:

The **setGov** function allows for setting a new admin account.

However, there is no check for the zero address and no two-step ownership transferral. As a result, if an incorrect address is passed into this function, the contract would be accidentally ossified.

```
function setGov(address _gov) external  {
  _onlyGov();
  gov = _gov;

}
```

QSWP-13. FUNCTIONS CAN BE MARKED AS EXTERNAL

SEVERITY: Low

PATH: see description

REMEDIATION: each function mentioned should be marked external instead of public

STATUS: fixed

DESCRIPTION:

We have identified the following locations where functions can be marked as **external** in order to save gas:

1. **oracle/pyth/PythPriceFeed.sol:initialize** (L87-102).

```
function initialize(uint256 _minAuthorizations, address[] memory _signers, address[] memory
_updaters) public onlyGov {
  [...]
}
```

QSWP-14. FUNCTION PARAMETERS CAN BE MARKED AS CALLDATA

SEVERITY: Low

PATH: see description

REMEDIATION: each parameter mentioned in the description should be marked as calldata instead of memory

STATUS: fixed

DESCRIPTION:

We have identified the following locations where the function parameters can be marked as **calldata** instead of **memory**, which will save gas on each function call:

1. **oracle/pyth/PythPriceFeed.sol:initialize** (L87-102) for the parameters **_signers** and **_updaters** (after the function is marked as **external**).
2. **oracle/pyth/PythPriceFeed.sol:setPriceFeedIds** (L144-149) for the parameters **_tokens** and **_priceFeedIds**.
3. **oracle/pyth/PythPriceFeed.sol:setPriceConfidenceMultipliers** (L144-149) for the parameters **_tokens** and **_priceConfidenceMultipliers**.
4. **oracle/pyth/PythPriceFeed.sol:setPriceConfidenceThresholds** (L144-149) for the parameters **_tokens** and **_priceConfidenceThresholds**.

```
function initialize(uint256 _minAuthorizations, address[] memory _signers, address[] memory
_updaters) public onlyGov {
  [..]
}

function setPriceFeedIds(address[] memory _tokens,  bytes32[] memory _priceFeedIds) external
onlyTokenManager {
  [..]
}

function setPriceConfidenceMultipliers(address[] memory _tokens,  uint256[] memory
_priceConfidenceMultipliers) external  onlyGov {
  [..]
}

function setPriceConfidenceThresholds(address[] memory _tokens,  uint256[] memory
_priceConfidenceThresholds) external  onlyGov {
  [..]
}
```

QSWP-15. VARIABLES ARE INITIALISED TO DEFAULT VALUES

SEVERITY: Low

PATH: see description

REMEDIATION: the variables mentioned in the description should have their initialisation removed, only the declaration is required

STATUS: fixed

DESCRIPTION:

We have identified the following locations where storage variables are initialised to their default values. This is unnecessary and costs gas.

1. **oracle/pyth/PythPriceFeed.sol:isSpreadEnabled** (L26).
2. **oracle/pyth/PythPriceFeed.sol:disableFastPriceVoteCount** (L41).

```
bool public isSpreadEnabled = false;

uint256 public disableFastPriceVoteCount = 0;
```

QSWP-16. EARLY EXIT OPTIMISATION FOR CONDITIONALS

SEVERITY: Low

PATH: see description

REMEDIATION: see description

STATUS: fixed

DESCRIPTION:

We have identified the following locations where conditionals can be optimised to early exit and save gas:

1. **oracle/pyth/PythPriceFeed.sol:getPrice** (L251-308), the variable **hasSpread** is set to the result of a conditional where the first element uses a function call with storage loads, while the second element is a simple arithmetic operation.

```
bool hasSpread = !favorFastPrice() || diffBasisPoints > maxDeviationBasisPoints;
```

Optimise the order for the conditionals.

For example:

```
bool hasSpread = diffBasisPoints > maxDeviationBasisPoints || !favorFastPrice();
```

QSWP-17. FAVOUR FAST PRICE CHECK CAN BE OPTIMISED

SEVERITY: Low

PATH: oracle/pyth/PythPriceFeed.sol:favorFastPrice:L310-321

REMEDIATION: see description

STATUS: fixed

DESCRIPTION:

The function favorFastPrice only returns booleans based on conditions, it can therefore be optimised into one evaluation:

```
function favorFastPrice() public view returns (bool) {
  return !isSpreadEnabled && disableFastPriceVoteCount < minAuthorizations;
}
```

```
function favorFastPrice() public view returns (bool) {
  if (isSpreadEnabled) {
    return false;
  }

  if (disableFastPriceVoteCount >= minAuthorizations) {
    // force a spread if watchers have flagged an issue with the fast price
    return false;
  }

  return true;
}
```

QSWP-18. REDUNDANT USAGE OF SAFEMATH

SEVERITY: Low

PATH: oracle/pyth/PythPriceFeed.sol

REMEDIATION: remove the import of OpenZeppelin's SafeMath library and replace all the arithmetic operations with their Solidity equivalent

STATUS: fixed

DESCRIPTION:

The **PythPriceFeed** imports and uses OpenZeppelin's SafeMath library for arithmetic operations. However, the contract uses Solidity **0.8.0**, which already checks arithmetic operations for over- and underflow. As a result, this is redundant and wastes gas on each function call containing these operations.

```
import "@openzeppelin/contracts/utils/math/SafeMath.sol";
[...]
contract PythPriceFeed is IPythPriceFeed, Governable {
  using SafeMath for uint256;
  [...]
}
```

QSWP-21. TIMELOCK LEVERAGE ENABLE/DISABLE GAS OPTIMISATION

SEVERITY: Low

PATH: Timelock.sol:enableLeverage, disableLeverage (L229-247, L249-267)

REMEDIATION: add an additional function to the Vault contract that allows for setting marginFeeBasisPoints only, so that this function can be used in Timelock for enableLeverage and disableLeverage

STATUS: fixed

DESCRIPTION:

These functions are called from the **BasePositionManager.sol:_increasePosition** and **_decreasePosition** before changing an order in the **Vault** contract.

Both function set the fees of the vault using **Timelock.setFees**, which calls **Vault.setFees**. However, only the **marginFeeBasisPoints** configuration variable is changed and 8 other variables are set to their original value twice, wasting gas.

```
function enableLeverage(address _vault) external override onlyHandlerAndAbove {
    IVault vault = IVault(_vault);

    if (shouldToggleIsLeverageEnabled) {
        vault.setIsLeverageEnabled(true);
    }

    vault.setFees(
        vault.taxBasisPoints(),
        vault.stableTaxBasisPoints(),
        vault.mintBurnFeeBasisPoints(),
        vault.swapFeeBasisPoints(),
        vault.stableSwapFeeBasisPoints(),
        marginFeeBasisPoints,
        vault.liquidationFeeUsd(),
        vault.minProfitTime(),
        vault.hasDynamicFees()
    );
}

function disableLeverage(address _vault) external override onlyHandlerAndAbove {
    IVault vault = IVault(_vault);

    if (shouldToggleIsLeverageEnabled) {
        vault.setIsLeverageEnabled(false);
    }

    vault.setFees(
        vault.taxBasisPoints(),
        vault.stableTaxBasisPoints(),
        vault.mintBurnFeeBasisPoints(),
        vault.swapFeeBasisPoints(),
        vault.stableSwapFeeBasisPoints(),
        maxMarginFeeBasisPoints, // marginFeeBasisPoints
        vault.liquidationFeeUsd(),
        vault.minProfitTime(),
        vault.hasDynamicFees()
    );
}
```

```
function setFees(
    uint256 _taxBasisPoints,
    uint256 _stableTaxBasisPoints,
    uint256 _mintBurnFeeBasisPoints,
    uint256 _swapFeeBasisPoints,
    uint256 _stableSwapFeeBasisPoints,
    uint256 _marginFeeBasisPoints,
    uint256 _liquidationFeeUsd,
    uint256 _minProfitTime,
    bool _hasDynamicFees
) external override {
    _onlyGov();
    _validate(_taxBasisPoints <= MAX_FEE_BASIS_POINTS, 3);
    _validate(_stableTaxBasisPoints <= MAX_FEE_BASIS_POINTS, 4);
    _validate(_mintBurnFeeBasisPoints <= MAX_FEE_BASIS_POINTS, 5);
    _validate(_swapFeeBasisPoints <= MAX_FEE_BASIS_POINTS, 6);
    _validate(_stableSwapFeeBasisPoints <= MAX_FEE_BASIS_POINTS, 7);
    _validate(_marginFeeBasisPoints <= MAX_FEE_BASIS_POINTS, 8);
    _validate(_liquidationFeeUsd <= MAX_LIQUIDATION_FEE_USD, 9);
    taxBasisPoints = _taxBasisPoints;
    stableTaxBasisPoints = _stableTaxBasisPoints;
    mintBurnFeeBasisPoints = _mintBurnFeeBasisPoints;
    swapFeeBasisPoints = _swapFeeBasisPoints;
    stableSwapFeeBasisPoints = _stableSwapFeeBasisPoints;
    marginFeeBasisPoints = _marginFeeBasisPoints;
    liquidationFeeUsd = _liquidationFeeUsd;
    minProfitTime = _minProfitTime;
    hasDynamicFees = _hasDynamicFees;
}
```

QSWP-3. ORDERBOOK DECREASEORDER EVENTS ARE MISSING VALUES

SEVERITY: Informational

PATH: core/OrderBook.sol:CreateDecreaseOrder, CancelDecreaseOrder, ExecuteDecreaseOrder, UpdateDecreaseOrder (L140-189)

REMEDIATION: add the missing fields

STATUS: fixed

DESCRIPTION:

The Quickswap changes have added the following fields to the **DecreaseOrder** struct: **receiveToken**, **minOut** and **withdrawETH**.

However, only **receiveToken** has been added to the **CreateDecreaseOrder** event. The other new fields are missing and the other events don't have any of the new fields.

```
event CreateDecreaseOrder(
    address indexed account,
    uint256 orderIndex,
    address collateralToken,
    address receiveToken,
    uint256 collateralDelta,
    address indexToken,
    uint256 sizeDelta,
    bool isLong,
    uint256 triggerPrice,
    bool triggerAboveThreshold,
    uint256 executionFee
);

event CancelDecreaseOrder(
    address indexed account,
    uint256 orderIndex,
    address collateralToken,
    uint256 collateralDelta,
    address indexToken,
    uint256 sizeDelta,
    bool isLong,
    uint256 triggerPrice,
    bool triggerAboveThreshold,
    uint256 executionFee
);
event ExecuteDecreaseOrder(
    address indexed account,
    uint256 orderIndex,
    address collateralToken,
    uint256 collateralDelta,
    address indexToken,
    uint256 sizeDelta,
    bool isLong,
    uint256 triggerPrice,
    bool triggerAboveThreshold,
    uint256 executionFee,
    uint256 executionPrice
);
event UpdateDecreaseOrder(
    address indexed account,
    uint256 orderIndex,
    address collateralToken,
    uint256 collateralDelta,
    address indexToken,
    uint256 sizeDelta,
    bool isLong,
    uint256 triggerPrice,
    bool triggerAboveThreshold
);
```

QSWP-5. ITERABLEMAPPING DOES NOT RETURN TRUE UPON SUCCESSFUL REMOVAL

SEVERITY: Informational

PATH: libraries/utils/IterableMapping.sol:remove:L37-44

REMEDIATION: add a return true statement at the end of the remove function

STATUS: fixed

DESCRIPTION:

The **remove** function for the Iterable Mapping library has an early exit with **return false**, however it is missing a final **return true** statement when the key has actually been removed. The defualt value of the **success** return parameter will be **false**, so the function will always return **false**.

```
function remove(itmap storage self, uint256 key) internal returns (bool success) {
    uint256 keyIndex = self.data[key].keyIndex;
    if (keyIndex == 0)
        return false;
    delete self.data[key];
    self.keys[keyIndex - 1].deleted = true;
    self.size --;
}
```

QSWP-7. MISSING EVENTS IN ORDERBOOK

SEVERITY: Informational

PATH: OrderBook.sol:setOrderExecutor, setMinExecutionFee, setChainlinkOrderExecutionActive, setMinPurchaseTokenAmountUsd, setGov

REMEDIATION: add events in each of the mentioned functions so state updates can be tracked off-chain

STATUS: fixed

DESCRIPTION:

All of mentioned functions update the state of the contract but they do not emit any events.

```
function setOrderExecutor(
  address _orderExecutor
)external  {
  _onlyGov();
  orderExecutor = _orderExecutor;

}
```

```
function setMinExecutionFee(uint256 _minExecutionFee) external {
  _onlyGov();
  minExecutionFee = _minExecutionFee;

}
function setChainlinkOrderExecutionActive(bool _chainlinkOrderExecutionActive) external {
  _onlyGov();
  chainlinkOrderExecutionActive = _chainlinkOrderExecutionActive;



}
function setMinPurchaseTokenAmountUsd(uint256 _minPurchaseTokenAmountUsd) external {
  _onlyGov();
  minPurchaseTokenAmountUsd = _minPurchaseTokenAmountUsd;

}

function setGov(address _gov) external {
  _onlyGov();
  gov = _gov;

}
```

QSWP-1. EVENTS ARE MISSING INDEXED TOPICS

SEVERITY: Informational

PATH:
core/PositionManager.sol:SetPartnerMinStayingOpenTime:L32

REMEDIATION: see description

STATUS: fixed

DESCRIPTION:

The event **SetPartnerMinStayingOpenTime** should add the **indexed** keyword to the account value, making it a searchable topic.

```
event SetPartnerMinStayingOpenTime(address account, uint256 minTime);
```

QSWP-8. NO SIGNER COUNT OR MINIMUM AUTHORISATIONS CHECK

SEVERITY: Informational

PATH:
oracle/pyth/PythPriceFeed.sol:setMinAuthorizations:L165-167

REMEDIATION: keep a counter that increases and decreases in setSigner correspondingly. Do make sure that there is an assertion that checks whether the user already had the status (so the counter cannot increase/decrease twice for the same user)

STATUS: fixed

DESCRIPTION:

Currently there is no counter to keep track of the number of signers. As a result, it is not possible to check the **minAuthorizations** variable against the total number of signers, to make sure that the threshold is not actually greater, making it impossible to disable fast price feed.

```
function setMinAuthorizations(uint256 _minAuthorizations) external onlyTokenManager {
  minAuthorizations = _minAuthorizations;
}
```

QSWP-9. ADDING/REMOVING A SIGNER/UPDATER IS UNTRACKABLE OFF-CHAIN

SEVERITY: Informational

PATH: oracle/pyth/PythPriceFeed.sol:setSigner, setUpdater (L107-109, L111-113)

REMEDIATION: at least emit events in both setSigner and setUpdater (as well as in initialize) so it is easy to keep track of changes off-chain

STATUS: fixed

DESCRIPTION:

The functions to add/remove a signer/updater do not emit events and there is also no list of these users.

As a result, it is very difficult to track which and how many signers/updaters there are.

```
function setSigner(address _account, bool _isActive) external override onlyGov {
  isSigner[_account] = _isActive;
}

function setUpdater(address _account, bool _isActive) external override onlyGov {
  isUpdater[_account] = _isActive;
}
```

QSWP-10. MISSING ARRAY PARAMETERS LENGTH CHECK

SEVERITY: Informational

PATH: see description

REMEDIATION: see description

STATUS: fixed

DESCRIPTION:

We have identified the following locations where there are 2 or more arrays as function parameters that require the same length but are missing these length checks:

1. oracle/pyth/PythPriceFeed.sol:setPriceFeedIds (L144-149)
2. oracle/pyth/PythPriceFeed.sol:setPriceConfidenceMultipliers (L151-156)
3. oracle/pyth/PythPriceFeed.sol:setPriceConfidenceThresholds (L158-163)
4. oracle/FastPriceFeed.sol:setMaxCumulativeDiffs (L198-203)

```
function setPriceFeedIds(address[] memory _tokens, bytes32[] memory _priceFeedIds) external onlyTokenManager {
  for (uint256 i = 0; i < _tokens.length; i++) {
    address token = _tokens[i];
    priceFeedIds[token] = _priceFeedIds[i];
  }
}

function setPriceConfidenceMultipliers(address[] memory _tokens, uint256[] memory _priceConfidenceMultipliers)
external onlyGov {
  for (uint256 i = 0; i < _tokens.length; i++) {
    address token = _tokens[i];
    priceConfidenceMultipliers[token] = _priceConfidenceMultipliers[i];
  }
}

function setPriceConfidenceThresholds(address[] memory _tokens, uint256[] memory _priceConfidenceThresholds)
external onlyGov {
  for (uint256 i = 0; i < _tokens.length; i++) {
    address token = _tokens[i];
    priceConfidenceThresholds[token] = _priceConfidenceThresholds[i];
  }
}

function setMaxCumulativeDeltaDiffs(address[] memory _tokens, uint256[] memory _maxCumulativeDeltaDiffs)
external override onlyTokenManager {
  for (uint256 i = 0; i < _tokens.length; i++) {
    address token = _tokens[i];
    maxCumulativeDeltaDiffs[token] = _maxCumulativeDeltaDiffs[i];
  }
}
```

Check the length of the input parameters so that the error is caught to improve user experience.

For example:

```
require(a1.length == a2.length, "Length mismatch");
```

QSWP-11. MISSING ADDRESS ZERO CHECKS

SEVERITY: Informational

PATH: see description

REMEDIATION: see description

STATUS: fixed

DESCRIPTION:

We have identified the following locations where there are missing address zero checks. If this were to happen, the functionality would break.

1. **oracle/pyth/PythPriceFeed.sol:constructor** (L70-85) for the parameters **_tokenManager**, **_positionRouter** and **_pythContract**.
2. **oracle/pyth/PythPriceFeed.sol:setTokenManager** (L136-138) for the parameter **_tokenManager**.
3. **oracle/pyth/PythPriceFeed.sol:setPositionRouter** (L104-106) for the parameter **_positionRouter**.

```
constructor(
 uint256 _priceDuration,
 uint256 _maxPriceUpdateDelay,
 uint256 _maxDeviationBasisPoints,
 address _tokenManager,
 address _positionRouter,
 address _pythContract
) {
  require(_priceDuration <= MAX_PRICE_DURATION, "FastPriceFeed: invalid _priceDuration");
  priceDuration = _priceDuration;
  maxPriceUpdateDelay = _maxPriceUpdateDelay;
  maxDeviationBasisPoints = _maxDeviationBasisPoints;
  tokenManager = _tokenManager;
  positionRouter = _positionRouter;
  pyth = IPyth(_pythContract);
}
function setPositionRouter(address _positionRouter) external onlyTokenManager {
  positionRouter = _positionRouter;
}
function setTokenManager(address _tokenManager) external onlyTokenManager {
  tokenManager = _tokenManager;
}
```

Add address zero checks for the locations and corresponding parameters.

For example:

```
require(a != address(0), "Address zero");
```