OMNISCIA

September 19, 2023

SMART CONTRACT AUDIT REPORT

Omniscia Kwenta Audit
Smart Margin V3


omniscia.io


info@omniscia.io


Online report: Available Here

Omniscia.io is one of the fastest growing and most trusted blockchain security firms and has rapidly become a true market leader. To date, our team has collectively secured over 370+ clients, detecting 1,500+ high-severity issues in widely adopted smart contracts.

Founded in France at the start of 2020, and with a track record spanning back to 2017, our team has been at the forefront of auditing smart contracts, providing expert analysis and identifying potential vulnerabilities to ensure the highest level of security of popular smart contracts, as well as complex and sophisticated decentralized protocols.

Our clients, ecosystem partners, and backers include leading ecosystem players such as L'Oréal, Polygon, AvaLabs, Gnosis, Morpho, Vesta, Gravita, Olympus DAO, Fetch.ai, and LimitBreak, among others.

To keep up to date with all the latest news and announcements follow us on twitter @omniscia_sec.

omniscia.io


info@omniscia.io

Smart Margin V3 Security Audit

Audit Report Revisions

Commit Hash	Date	Audit Report Hash
99f3dc5ab8	September 11th 2023	f5ad8ac483
711b7950d3	September 19th 2023	c71044fb87

Audit Overview

We were tasked with performing an audit of the Kwenta codebase and in particular their Smart Margin V3 protocol that closely integrates with the Synthetix V3 protocol.

Over the course of the audit, we identified a significant flaw in the cryptographic nonce system utillized by the `Engine` that rendered it trivial to perform a Denial-of-Service attack on all conditional orders.

We advise the Kwenta team to closely evaluate all minor-and-above findings identified in the report and promptly remediate them as well as consider all optimizational exhibits identified in the report.

Post-Audit Conclusion

The Kwenta team iterated through all findings within the report and provided us with a revised commit hash to evaluate all exhibits on.

We evaluated all alleviations performed by Kwenta and have identified that all exhibits have been adequately dealt with no outstanding issues remaining in the report.

Contracts Assessed

Files in Scope	Repository	Commit(s)
ConditionalOrderHashLib.sol (COH)	smart-margin-v3	99f3dc5ab8, 711b7950d3
EIP712.sol (EIP)	smart-margin-v3	99f3dc5ab8, 711b7950d3
Engine.sol (EEN)	smart-margin-v3	99f3dc5ab8, 711b7950d3
ERC721Receivable.sol (ERC)	smart-margin-v3	99f3dc5ab8, 711b7950d3
MathLib.sol (MLB)	smart-margin-v3	99f3dc5ab8, 711b7950d3
Multicallable.sol (MEL)	smart-margin-v3	99f3dc5ab8, 711b7950d3
SignatureCheckerLib.sol (SCL)	smart-margin-v3	99f3dc5ab8, 711b7950d3

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	1	1	0	0
Informational	8	6	0	2
Minor	5	4	0	1
Medium	3	3	0	0
Major	0	0	0	0

During the audit, we filtered and validated a total of **3 findings utilizing static analysis** tools as well as identified a total of **14 findings during the manual review** of the codebase. We strongly recommend that any minor severity or higher findings are dealt with promptly prior to the project's launch as they can introduce potential misbehaviours of the system as well as exploits.

Smart Margin V3 Security Audit

Audit Report Revisions

Commit Hash	Date	Audit Report Hash
99f3dc5ab8	September 11th 2023	f5ad8ac483
711b7950d3	September 19th 2023	c71044fb87

Audit Overview

We were tasked with performing an audit of the Kwenta codebase and in particular their Smart Margin V3 protocol that closely integrates with the Synthetix V3 protocol.

Over the course of the audit, we identified a significant flaw in the cryptographic nonce system utillized by the `Engine` that rendered it trivial to perform a Denial-of-Service attack on all conditional orders.

We advise the Kwenta team to closely evaluate all minor-and-above findings identified in the report and promptly remediate them as well as consider all optimizational exhibits identified in the report.

Post-Audit Conclusion

The Kwenta team iterated through all findings within the report and provided us with a revised commit hash to evaluate all exhibits on.

We evaluated all alleviations performed by Kwenta and have identified that all exhibits have been adequately dealt with no outstanding issues remaining in the report.

Contracts Assessed

Files in Scope	Repository	Commit(s)
ConditionalOrderHashLib.sol (COH)	smart-margin-v3	99f3dc5ab8, 711b7950d3
EIP712.sol (EIP)	smart-margin-v3	99f3dc5ab8, 711b7950d3
Engine.sol (EEN)	smart-margin-v3	99f3dc5ab8, 711b7950d3
ERC721Receivable.sol (ERC)	smart-margin-v3	99f3dc5ab8, 711b7950d3
MathLib.sol (MLB)	smart-margin-v3	99f3dc5ab8, 711b7950d3
Multicallable.sol (MEL)	smart-margin-v3	99f3dc5ab8, 711b7950d3
SignatureCheckerLib.sol (SCL)	smart-margin-v3	99f3dc5ab8, 711b7950d3

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	1	1	0	0
Informational	8	6	0	2
Minor	5	4	0	1
Medium	3	3	0	0
Major	0	0	0	0

During the audit, we filtered and validated a total of **3 findings utilizing static analysis** tools as well as identified a total of **14 findings during the manual review** of the codebase. We strongly recommend that any minor severity or higher findings are dealt with promptly prior to the project's launch as they can introduce potential misbehaviours of the system as well as exploits.

Compilation

The project utilizes `foundry` as its development pipeline tool, containing an array of tests and scripts coded in Solidity.

To compile the project, the `build` command needs to be issued via the `forge` CLI tool:

```bash
forge build
```

The `forge` tool automatically selects Solidity version `0.8.18` based on the `pragma` versions of the contracts as the `foundry.toml` file does not specify a version explicitly.

The project does not contain any discrepancies in regard to the Solidity versions as all contracts have been locked to a specific version.

In detail, all contracts have been locked to `0.8.18`, the same version utilized for our static analysis as well as optimizational review of the codebase.

During compilation with the `foundry` pipeline, no errors were identified that relate to the syntax or bytecode size of the contracts.

Static Analysis

The execution of our static analysis toolkit identified **21 potential issues** within the codebase of which **13 were ruled out to be false positives** or negligible findings.

The remaining **8 issues** were validated and grouped and formalized into the **3 exhibits** that follow:

ID	Severity	Addressed	Title
EEN-01S	Informational	Acknowledged	Illegible Numeric Value Representations
EEN-02S	Minor	Yes	Inexistent Sanitization of Input Addresses
EEN-03S	Medium	Nullified	Improper Invocations of EIP-20 `transfer` / `transferFrom`

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in Kwenta's Smart Margin V3 protocol.

As the project at hand implements a Synthetix V3 perpetual account manager, intricate care was put into ensuring that the **flow of funds & assets within the system conforms to the specifications and restrictions** laid forth within the protocol's specification.

We validated that **all state transitions of the system occur within sane criteria** and that all rudimentary formulas within the system execute as expected. We **pinpointed two important vulnerabilities** within the system which could have had **moderate ramifications** to its overall operation; we urge the Kwenta team to promptly evaluate and remediate them.

Additionally, the system was investigated for any other commonly present attack vectors such as re-entrancy attacks, mathematical truncations, logical flaws and **ERC / EIP** standard inconsistencies. The documentation of the project was satisfactory to a great extent, containing extensive in-line comments as well as repository-level documentation that greatly aided in achieving an understanding of the system.

A total of **14 findings** were identified over the course of the manual review of which **7 findings** concerned the behaviour and security of the system. The non-security related findings, such as optimizations, are included in the separate **Code Style** chapter.

The finding table in the ensuing page enumerates all the security / behavioural findings.

ID	Severity	Addressed	Title
COH-01M	Minor	Yes	Discrepancy of Type-Hashes
EEN-01M	Minor	Yes	Inexistent Validation of Confidence Interval
EEN-02M	Minor	Acknowledged	Inexistent Validation of Order Configuration
EEN-03M	Minor	Yes	Potentially Insecure Account Creation Workflow
EEN-04M	Medium	Yes	Improperly Shared Nonce System
EEN-05M	Medium	Yes	Potentially Insecure Order Consumption
SCL-01M	Unknown	Yes	Usage of Experimental Library

Code Style

During the manual portion of the audit, we identified **7 optimizations** that can be applied to the codebase that will decrease the operational cost associated with the execution of a particular function and generally ensure that the project complies with the latest best practices and standards in Solidity.

Additionally, this section of the audit contains any opinionated adjustments we believe the code should make to make it more legible as well as truer to its purpose.

These optimizations are enumerated below:

ID	Severity	Addressed	Title
ERC-01C	Informational	Nullified	Specification of Abstract Contract
EEN-01C	Informational	Yes	Inefficient Memory Usage
EEN-02C	Informational	Yes	Inefficient `if` Structure
EEN-03C	Informational	Yes	Non-Uniform Invocation Style
EEN-04C	Informational	Yes	Redundant Ternary Operator
MLB-01C	Informational	Acknowledged	Memory-Safe Optimization
MLB-02C	Informational	Yes	Optimization of Absolute Value Evaluation

Engine Static Analysis Findings

EEN-01S: Illegible Numeric Value Representations

Type	Severity	Location
Code Style	Informational	Engine.sol:L50, L53

Description:

The linked representations of numeric literals are sub-optimally represented decreasing the legibility of the codebase.

Example:

src/Engine.sol

SOL

```sol
50  uint256 public constant FEE_SCALING_FACTOR = 1000;
```

Recommendation:

To properly illustrate each value's purpose, we advise the following guidelines to be followed. For values meant to depict fractions with a base of `1e18`, we advise fractions to be utilized directly (i.e. `1e17` becomes `0.1e18`) as they are supported. For values meant to represent a percentage base, we advise each value to utilize the underscore (`_`) separator to discern the percentage decimal (i.e. `10000` becomes `100_00`, `300` becomes `3_00` and so on). Finally, for large numeric values we simply advise the underscore character to be utilized again to represent them (i.e. `1000000` becomes `1_000_000`).

Alleviation:

The Kwenta team evaluated this exhibit but opted to retain the codebase as is, acknowledging it.

EEN-02S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Minor	Engine.sol:L92-L102

Description:

The linked function(s) accept `address` arguments yet do not properly sanitize them.

Impact:

The presence of zero-value addresses, especially in `constructor` implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

Example:

src/Engine.sol

SOL

```sol
92  constructor(
93      address _perpsMarketProxy,
94      address _spotMarketProxy,
95      address _sUSDProxy,
96      address _oracle
97  ) {
98      PERPS_MARKET_PROXY = IPerpsMarketProxy(_perpsMarketProxy);
99      SPOT_MARKET_PROXY = ISpotMarketProxy(_spotMarketProxy);
100     SUSD = IERC20(_sUSDProxy);
101     ORACLE = IPyth(_oracle);
102 }
```

Recommendation:

We advise some basic sanitization to be put in place by ensuring that each `address` specified is non-zero.

Alleviation:

All four input arguments of the **`Engine::constructor`** are now sanitized via dedicated `if-revert` patterns.

While the exhibit is considered alleviated, we would advise all `if` conditionals to be merged into one as the `revert` reason is the same across them.

EEN-03S: Improper Invocations of EIP-20 `transfer` / `transferFrom`

Type	Severity	Location
Standard Conformity	Medium	Engine.sol:L205, L219

Description:

The linked statements do not properly validate the returned `bool` values of the **EIP-20** standard `transfer` & `transferFrom` functions. As the **standard dictates**, callers **must not** assume that `false` is never returned.

Impact:

If the code mandates that the returned `bool` is `true`, this will cause incompatibility with tokens such as USDT / Tether as no such `bool` is returned to be evaluated causing the check to fail at all times. On the other hand, if the token utilized can return a `false` value under certain conditions but the code does not validate it, the contract itself can be compromised as having received / sent funds that it never did.

Example:

src/Engine.sol

SOL

```
205 _synth.transferFrom(_from, address(this), uint256(_amount));
```

Recommendation:

Since not all standardized tokens are **EIP-20** compliant (such as Tether / USDT), we advise a safe wrapper library to be utilized instead such as `SafeERC20` by OpenZeppelin to opportunistically validate the returned `bool` only if it exists in each instance.

Alleviation:

The Kwenta team evaluated this exhibit and specified that the `Engine` contract is meant to interact with Synthetix synth-assets exclusively which `revert` appropriately rather than yield a `bool` when they face an error during a transfer.

Based on this fact we consider this exhibit nullified as the assets interacted with will behave correctly even without checking their return `bool` values.

ConditionalOrderHashLib Manual Review Findings

COH-01M: Discrepancy of Type-Hashes

Type	Severity	Location
Standard Conformity	Minor	ConditionalOrderHashLib.sol:L12, L17

Description:

The `_ORDER_DETAILS_TYPEHASH` will contain more variables declared at its end than the `_CONDITIONAL_ORDER_TYPEHASH` entry of the same name. While based on typehashes alone this is inconsequential, the `ConditionalOrderHashLib::hash(IEngine.ConditionalOrder memory)` function is utilizing the `_ORDER_DETAILS_TYPEHASH` methodology for hashing the payload, leading to an improperly declared `_CONDITIONAL_ORDER_TYPEHASH` type-hash.

Impact:

Calculations of the relevant type-hashes for the structures may end up being discrepant in relation to off-chain calculations due to the incorrect type-hashes specified.

Example:

src/libraries/ConditionalOrderHashLib.sol

SOL

```sol
/// @notice pre-computed keccak256(OrderDetails struct)
bytes32 public constant _ORDER_DETAILS_TYPEHASH = keccak256(
    "OrderDetails(uint128 marketId,uint128 accountId,int128 sizeDelta,uint128 settlementStrategyId,uint256 acceptablePrice,bool isReduceOnly,bytes32 trackingCode,address referrer)"
);

/// @notice pre-computed keccak256(ConditionalOrder struct)
bytes32 public constant _CONDITIONAL_ORDER_TYPEHASH = keccak256(
    "ConditionalOrder(OrderDetails orderDetails,address signer,uint128 nonce,bool requireVerified,address trustedExecutor,bytes[] conditions)OrderDetails(uint128 marketId,uint128 accountId,int128 sizeDelta,uint128 settlementStrategyId,uint256 acceptablePrice)"
);

/// @notice hash the OrderDetails struct
/// @param orderDetails OrderDetails struct
/// @return hash of the OrderDetails struct
function hash(IEngine.OrderDetails memory orderDetails)
    internal
    pure
    returns (bytes32)
{
    return keccak256(
        abi.encode(
            _ORDER_DETAILS_TYPEHASH,
            orderDetails.marketId,
            orderDetails.accountId,
            orderDetails.sizeDelta,
            orderDetails.settlementStrategyId,
            orderDetails.acceptablePrice,
            orderDetails.isReduceOnly,
            orderDetails.trackingCode,
            orderDetails.referrer
        )
```

SOL

```sol
    );
}

/// @notice hash the ConditionalOrder struct
/// @param co ConditionalOrder struct
/// @return hash of the ConditionalOrder struct
function hash(IEngine.ConditionalOrder memory co)
    internal
    pure
    returns (bytes32)
{
    bytes32 orderDetailsHash = hash(co.orderDetails);
    return keccak256(
        abi.encode(
            _CONDITIONAL_ORDER_TYPEHASH,
            orderDetailsHash,
            co.signer,
            co.nonce,
            co.requireVerified,
            co.trustedExecutor,
            co.conditions
        )
    );
}
```

Recommendation:

We advise the type-hash of conditional orders to be corrected, properly depicting all variables included in the `_ORDER_DETAILS_TYPEHASH` and specifically the `isReduceOnly`, `trackingCode`, and `referrer` variables. We would like to note that even if they are meant to be zero, they still are part of the hashing methodology employed by `_ORDER_DETAILS_TYPEHASH` and should be specified regardless.

Alleviation:

All three variables that were missing from the `OrderDetails` structure defined in `_CONDITIONAL_ORDER_TYPEHASH` have been introduced, standardizing its representation and thus alleviating this exhibit.

Engine Manual Review Findings

EEN-01M: Inexistent Validation of Confidence Interval

Type	Severity	Location
External Call Validation	Minor	Engine.sol:L508, L522

Description:

The `Engine::isPriceAbove` and `Engine::isPriceBelow` function implementations do not apply any restriction on the confidence interval yielded by the Pyth Network. As such, it is possible for the conditions to incorrectly validate in volatile market conditions.

Impact:

While the severity of this exhibit is significant and would cause incorrect prices to be consumed, it would only manifest in volatile market conditions. As such, we consider this exhibit to be of "minor" severity.

Example:

src/Engine.sol

SOL

```sol
499 /// @inheritdoc IEngine
500 function isPriceAbove(bytes32 _assetId, int64 _price)
501     public
502     view
503     override
504     returns (bool)
505 {
506     /// @dev reverts if the price has not been updated
507     /// within the last `getValidTimePeriod()` seconds
508     PythStructs.Price memory priceData = ORACLE.getPrice(_assetId);
509
510     return priceData.price > _price;
511 }
512
513 /// @inheritdoc IEngine
514 function isPriceBelow(bytes32 _assetId, int64 _price)
515     public
516     view
517     override
518     returns (bool)
519 {
520     /// @dev reverts if the price has not been updated
521     /// within the last `getValidTimePeriod()` seconds
522     PythStructs.Price memory priceData = ORACLE.getPrice(_assetId);
523
524     return priceData.price < _price;
525 }
```

Recommendation:

We advise the functions to accept an additional argument that specifies the maximum confidence interval permitted (as a lower one would indicate greater accuracy), ensuring that the `priceData` values consumed are done so in relatively stable market conditions controlled by the condition's creator.

Alleviation:

Both functions have been updated to accept a `_confidenceInterval` argument that enables a certain degree of validation over the asset's volatility, alleviating this exhibit in full.

EEN-02M: Inexistent Validation of Order Configuration

Type	Severity	Location
Input Sanitization	Minor	Engine.sol:L402-L411

Description:

Based on the implementation of the `Engine::canExecute` function, a `ConditionalOrder` condition should either have a set of `conditions` **OR** a `trustedExecutor` but not both.

Impact:

Misconfigured `ConditionalOrder` payloads can lead to successful validations of orders that do not have any conditions or have a specific trusted executor alongside conditions.

Example:

src/Engine.sol

SOL

```sol
function canExecute(
    ConditionalOrder calldata _co,
    bytes calldata _signature
) public view override returns (bool) {
    // verify nonce has not been executed before
    if (executedOrders[_co.nonce]) return false;

    // verify signer is authorized to interact with the account
    if (!verifySigner(_co)) return false;

    // verify signature is valid for signer and order
    if (!verifySignature(_co, _signature)) return false;

    // verify conditions are met
    if (_co.requireVerified) {
        // if the order requires verification, then all conditions
        // defined by "conditions" for the order must be met
        if (!verifyConditions(_co)) return false;
    } else {
        // if the order does not require verification, then the caller
        // must be the trusted executor defined by "trustedExecutor"
        if (msg.sender != _co.trustedExecutor) return false;
    }

    return true;
}
```

Recommendation:

We advise the code to properly ensure that in case `requireVerified` has been specified, a non-zero number of `conditions` have been specified and that the `trustedExecutor` is the zero-address, and in the case of `!requireVerified` that the `conditions` specified have a `length` of `0`.

Alleviation:

The Kwenta team evaluated this exhibit and stated that it is the responsibility of the off-chain actor to specify properly configured conditions. As such, we consider this exhibit acknowledged.

EEN-03M: Potentially Insecure Account Creation Workflow

Type	Severity	Location
Logical Fault	Minor	Engine.sol:L109-L115

Description:

The `Engine` contract will inherit from the `ERC721Receivable` implementation to allow it to receive **EIP-721** assets from arbitrary tokens and senders in order to properly handle the perpetual market account NFTs that are created via the **`Engine::createAccount`** function.

Impact:

It is presently possible for NFTs to be accidentally sent to the `Engine` contract and be irrecoverable. We consider this case to not be rare as the contract is meant to handle Synthetix V3 perpetual accounts and users may incorrectly send their NFT instead of assigning the corresponding Synthetix V3 permissions to the `Engine` contract.

Example:

src/Engine.sol

SOL

```sol
109 function createAccount() external override returns (uint128 accountId) {
110     accountId = PERPS_MARKET_PROXY.createAccount();
111
112     IERC721 accountNftToken =
113         IERC721(PERPS_MARKET_PROXY.getAccountTokenAddress());
114     accountNftToken.safeTransferFrom(address(this), msg.sender, accountId);
115 }
```

Recommendation:

Given that the `ERC721Receivable` permits any **EIP-721** asset to be received, it is presently possible for mis-sent NFTs to be permanently locked in the `Engine` contract.

As such, we advise the **`Engine::createAccount`** function to simply store the `msg.sender` to a storage-level variable that will consequently be utilized by the **`ERC721Receivable::onERC721Received`** hook to relay the received NFT to the caller of the **`Engine::createAccount`** function.

This adjustment would ensure that accidental NFT transfers are impossible as the temporary storage variable would be assigned before the `PERPS_MARKET_PROXY::createAccount` call and be deleted after it, causing **`ERC721Receivable::onERC721Received`** function invocations to revert if the storage variable is zero.

Alleviation:

As a response to this exhibit the Kwenta team entirely removed the functionality to create an account. As a result, we consider this exhibit alleviated indirectly.

EEN-04M: Improperly Shared Nonce System

Type	Severity	Location
Logical Fault	Medium	Engine.sol:L320

Description:

The `Engine::execute` function will utilize a shared `nonce` system regardless of which `accountId` is being operated on. As a result, it is possible to perform a denial-of-service attack trivially by submitting a no-op `Engine::execute` payload with the `nonce` of a to-be-processed `Engine::execute` transaction.

Impact:

All `Engine::execute` transactions can be hijacked as a result of a denial-of-service attack trivially in the current implementation due to a global nonce system.

Example:

src/Engine.sol

SOL

```sol
304 function execute(ConditionalOrder calldata _co, bytes calldata _signature)
305     external
306     override
307     returns (
308         IPerpsMarketProxy.Data memory retOrder,
309         uint256 fees,
310         uint256 conditionalOrderFee
311     )
312 {
313     /// @dev check: (1) nonce has not been executed before
314     /// @dev check: (2) signer is authorized to interact with the account
315     /// @dev check: (3) signature for the order was signed by the signer
316     /// @dev check: (4) conditions are met || trusted executor is msg sender
317     if (!canExecute(_co, _signature)) revert CannotExecuteOrder();
318
319     /// @dev mark the nonce as executed; prevents replay
320     executedOrders[_co.nonce] = true;
```

Recommendation:

We advise the code to associate the `nonce` consumed with the `accountId` being utilized, ensuring that nonces are tied per account and are not global across all users interacting with the `Engine` contract.

Alleviation:

Nonces are now consumed and handled on a per account ID basis, using a specialized bitmap to allow a practically limitless number of unordered nonces.

Given that nonces are no longer shared between accounts, we consider this exhibit fully alleviated.

EEN-05M: Potentially Insecure Order Consumption

Type	Severity	Location
Logical Fault	Medium	Engine.sol:L345-L347

Description:

The `Engine::execute` function is meant to be utilized alongside signed payloads to execute a set of actions against a particular market and perpetual position.

While the system does expose certain conditions as part of the `ConditionalOrder` structure, they are rudimentary and in certain cases insufficient. The reason behind this is that the system makes certain assumptions (i.e. automatically setting `sizeDelta` and `conditionalOrderFee`) which are not part of the signed payload.

Impact:

It is presently impossible to control the `sizeDelta` (under certain cases) and `conditionalOrderFee` that will be applied in the execution of a particular order due to the absence of important condition implementations.

Example:

src/Engine.sol

SOL

```sol
function execute(ConditionalOrder calldata _co, bytes calldata _signature)
    external
    override
    returns (
        IPerpsMarketProxy.Data memory retOrder,
        uint256 fees,
        uint256 conditionalOrderFee
    )
{
    /// @dev check: (1) nonce has not been executed before
    /// @dev check: (2) signer is authorized to interact with the account
    /// @dev check: (3) signature for the order was signed by the signer
    /// @dev check: (4) conditions are met || trusted executor is msg sender
    if (!canExecute(_co, _signature)) revert CannotExecuteOrder();

    /// @dev mark the nonce as executed; prevents replay
    executedOrders[_co.nonce] = true;

    /// @notice get size delta from order details
    /// @dev up to the caller to not waste gas by passing in a size delta of zero
    int128 sizeDelta = _co.orderDetails.sizeDelta;

    /// @notice handle reduce only orders
    if (_co.orderDetails.isReduceOnly) {
        (,, int128 positionSize) = PERPS_MARKET_PROXY.getOpenPosition(
            _co.orderDetails.accountId, _co.orderDetails.marketId
        );

        // ensure position exists; reduce only orders cannot increase position
size
        if (positionSize == 0) {
```

SOL

```
334             return (retOrder, 0, 0);
335         }
336
337         // ensure incoming size delta is NOT the same sign; i.e. reduce only
orders cannot increase position size
338         if (positionSize.isSameSign(sizeDelta)) {
339             return (retOrder, 0, 0);
340         }
341
342         // ensure incoming size delta is not larger than current position size
343         /// @dev reduce only orders can only reduce position size (i.e. approach
size of zero) and
344         /// cannot cross that boundary (i.e. short -> long or long -> short)
345         if (sizeDelta.abs128() > positionSize.abs128()) {
346             sizeDelta = -positionSize;
347         }
348     }
349
350     /// @dev fetch estimated order fees to be used to
351     /// calculate conditional order fee
352     (uint256 orderFees,) = PERPS_MARKET_PROXY.computeOrderFees({
353         marketId: _co.orderDetails.marketId,
354         sizeDelta: sizeDelta
355     });
356
357     /// @dev calculate conditional order fee based on scaled order fees
358     conditionalOrderFee = (orderFees * FEE_SCALING_FACTOR) / MAX_BPS;
359
360     /// @dev ensure conditional order fee is within bounds
361     if (conditionalOrderFee < LOWER_FEE_CAP) {
362         conditionalOrderFee = LOWER_FEE_CAP;
363     } else if (conditionalOrderFee > UPPER_FEE_CAP) {
364         conditionalOrderFee = UPPER_FEE_CAP;
365     }
```

Recommendation:

We advise the code to expose two more conditions, permitting the caller to validate the following conditions:

The introduction of these two conditions will permit significantly greater granularity in how conditional orders are restricted and will ensure that the `sizeDelta` and `conditionalOrderFee` calculations performed by the **`Engine::execute`** function can be validated in some way before an order is executed.

Alleviation:

Three conditions were introduced as part of alleviations for this exhibit that allow the position size to be validated in either an "above" or "below" style evaluation as well as the order fee to be validated in a "below" style evaluation.

Coupled with comments introduced in **`Engine::execute`** that detail why using the newly introduced conditions is sufficient in controlling the expected fee of the executed order, we consider this exhibit to be fully alleviated.

SignatureCheckerLib Manual Review Findings

SCL-01M: Usage of Experimental Library

Type	Severity	Location
Standard Conformity	Unknown	SignatureCheckerLib.sol:L22-L85

Description:

The `SignatureCheckerLib` implementation represents a version of the homonym contract from the `solady` repository.

The `solady` dependency tree is considered experimental. As a tangible example, **PR#554** was recently merged that introduced additional fixes as well as optimizations for the library, rendering the version used by Kwenta outdated and potentially insecure.

Impact:

While the current version of the code lacks the latest fixes as well as optimizations, the impact of the exhibit cannot be assessed as the fixes pertained to edge-cases in relation to the output of the `ecrecover` precompile.

Example:

src/libraries/SignatureCheckerLib.sol

SOL

```sol
if eq(signature.length, 65) {
    mstore(m, hash)
    mstore(
        add(m, 0x20),
        byte(0, calldataload(add(signature.offset, 0x40)))
    ) // `v`.
    calldatacopy(add(m, 0x40), signature.offset, 0x40) // `r`, `s`.
    pop(
        staticcall(
            gas(), // Amount of gas left for the transaction.
            1, // Address of `ecrecover`.
            m, // Start of input.
            0x80, // Size of input.
            m, // Start of output.
            0x20 // Size of output.
        )
    )
    // `returndatasize()` will be `0x20` upon success, and `0x00` otherwise.
    if mul(eq(mload(m), signer), returndatasize()) {
        isValid := 1
        break
    }
}
```

Recommendation:

We advise the code to utilize a different library than the `SignatureCheckerLib` for validating ECDSA signatures. Alternatively, if gas cost is of the essence, we advise the latest version of `solady` to be ported to the codebase and utilized as the current version lacks the latest fixes and optimizations.

Alleviation:

The latest version of the library from the `solady` repository has been imported and is now utilized within the Kwenta repository, alleviating this exhibit.

ERC721Receivable Code Style Findings

ERC-01C: Specification of Abstract Contract

Type	Severity	Location
Code Style	Informational	ERC721Receivable.sol:L6

Description:

The `ERC721Receivable` contract is not meant to be deployed as a standalone contract and is expected to be inherited by derivative contracts.

Example:

src/utils/ERC721Receivable.sol

SOL

```sol
contract ERC721Receivable {
    function onERC721Received(address, address, uint256, bytes calldata)
        external
        pure
        returns (bytes4)
    {
        return ERC721Receivable.onERC721Received.selector;
    }
}
```

Recommendation:

We advise the contract to be marked as `abstract`, optimizing the repository's code style.

Alleviation:

The contract is no longer present in the codebase rendering this exhibit no longer applicable.

Engine Code Style Findings

EEN-01C: Inefficient Memory Usage

Type	Severity	Location
Gas Optimization	Informational	Engine.sol:L442-L444, L446

Description:

The `Engine::verifySignature` function will declare a local variable `isValid` that will be immediately returned in the next statement.

Example:

src/Engine.sol

SOL

```sol
/// @inheritdoc IEngine
function verifySignature(
    ConditionalOrder calldata _co,
    bytes calldata _signature
) public view override returns (bool) {
    bool isValid = _signature.isValidSignatureNowCalldata(
        _hashTypedData(_co.hash()), _co.signer
    );

    return isValid;
}
```

Recommendation:

We advise the local variable declaration to be omitted and the `SignatureCheckerLib::isValidSignatureNowCalldata` function result to be returned to the caller directly, optimizing the function's gas cost.

Alleviation:

The result of the `SignatureCheckerLib::isValidSignatureNowCalldata` function is immediately yielded by the `Engine::verifySignature` function, optimizing its gas cost.

EEN-02C: Inefficient `if` Structure

Type	Severity	Location
Gas Optimization	Informational	Engine.sol:L427-L432, L434

Description:

The `Engine::verifySigner` function contains an `if` clause that will yield a value of `true` to the caller and an unwrapped statement that will yield `false` to the caller in case the `if` statement is not entered.

Example:

src/Engine.sol

SOL

```sol
420 /// @inheritdoc IEngine
421 function verifySigner(ConditionalOrder calldata _co)
422     public
423     view
424     override
425     returns (bool)
426 {
427     if (
428         isAccountOwner(_co.orderDetails.accountId, _co.signer)
429             || isAccountDelegate(_co.orderDetails.accountId, _co.signer)
430     ) {
431         return true;
432     }
433
434     return false;
435 }
```

Recommendation:

We advise the code to immediately yield the result of the `if` conditional, optimizing the function's gas cost.

Alleviation:

The result of the newly introduced `Engine::_isAccountOwnerOrDelegate` function is immediately yielded by the **`Engine::verifySigner`** function, optimizing its gas cost.

EEN-03C: Non-Uniform Invocation Style

Type	Severity	Location
Code Style	Informational	Engine.sol:L261-L269, L352-L355, L368-L374

Description:

The referenced functions are invoked utilizing the key-value argument declaration style whilst all other invocations utilize the index-based argument declaration style.

Example:

src/Engine.sol

SOL

```sol
367 /// @dev withdraw conditional order fee from account prior to executing order
368 _withdrawCollateral({
369     _to: msg.sender,
370     _synth: SUSD,
371     _accountId: _co.orderDetails.accountId,
372     _synthMarketId: USD_SYNTH_ID,
373     _amount: -int256(conditionalOrderFee)
374 });
375
376 /// @dev execute the order
377 (retOrder, fees) = _commitOrder(
378     _co.orderDetails.marketId,
379     _co.orderDetails.accountId,
380     sizeDelta,
381     _co.orderDetails.settlementStrategyId,
382     _co.orderDetails.acceptablePrice,
383     _co.orderDetails.trackingCode,
384     _co.orderDetails.referrer
385 );
```

Recommendation:

We advise the code to apply a single, uniform argument declaration style across all function invocations to ensure consistency in the code's style.

Alleviation:

The `Engine::_commitOrder` function call was updated to use the key-value declaration style, streamlining the invocation style of internal functions across the codebase as advised.

EEN-04C: Redundant Ternary Operator

Type	Severity	Location
Gas Optimization	Informational	Engine.sol:L535

Description:

The `Engine::isMarketOpen` function will evaluate whether the `PERPS_MARKET_PROXY::getMaxMarketSize` function result is `0` in a ternary operator and yield `false` if it is and `true` if it is not.

Example:

src/Engine.sol

SOL

```sol
527 /// @inheritdoc IEngine
528 function isMarketOpen(uint128 _marketId)
529     public
530     view
531     override
532     returns (bool)
533 {
534     return
535         PERPS_MARKET_PROXY.getMaxMarketSize(_marketId) == 0 ? false : true;
536 }
```

Recommendation:

We advise the code to immediately return the result of an inequality comparison with `0` rather than using a ternary operator to calculate the same result inefficiently.

Alleviation:

The result of an inequality check with `0` is immediately yielded by the relocated **`Engine::isMarketOpen`** function, optimizing its gas cost.

MathLib Code Style Findings

MLB-01C: Memory-Safe Optimization

Type	Severity	Location
Gas Optimization	Informational	MathLib.sol:L13, L42

Description:

The referenced `assembly` blocks purely operate on input arguments and statically-sized members, permitting the `("memory-safe")` attribute of the `assembly` block to be specified.

Example:

src/libraries/MathLib.sol

SOL

```sol
function abs128(int128 x) internal pure returns (uint128 z) {
    assembly {
        /// shl(128, x):
        /// shifts the number x to the left by 128 bits
        let y := shl(128, x)

        /// shr(255, x):
        /// shifts the number x to the right by 255 bits:
        /// IF the number is negative, the leftmost bit (bit 255) will be 1
        /// IF the number is positive,the leftmost bit (bit 255) will be 0

        /// sub(0, shr(255, x)):
        /// creates a mask of all 1s if x is negative
        /// creates a mask of all 0s if x is positive
        let mask := sub(0, shr(255, y))

        /// If x is negative, this effectively negates the number
        // if x is positive, it leaves the number unchanged, thereby computing the absolute value
        z := xor(mask, add(mask, y))

        /// shr(128, z):
        /// shifts the number z to the right by 128 bits after computing the absolute value
        z := shr(128, z)
    }
}
```

Recommendation:

We advise it to be done so, increasing the level of optimization the compiler can apply to these `assembly` blocks.

Alleviation:

The Kwenta team evaluated this exhibit but opted to retain the codebase as is, acknowledging it.

MLB-02C: Optimization of Absolute Value Evaluation

Type	Severity	Location
Gas Optimization	Informational	MathLib.sol:L16, L26, L30, L34

Description:

The `MathLib::abs128` function is meant to yield the absolute value of a signed integer 128-bits in length.

Example:

src/libraries/MathLib.sol

SOL

```sol
/// @notice get absolute value of the input, returned as an unsigned number.
/// @param x signed number
/// @return z uint128 absolute value of x
function abs128(int128 x) internal pure returns (uint128 z) {
    assembly {
        /// shl(128, x):
        /// shifts the number x to the left by 128 bits
        let y := shl(128, x)

        /// shr(255, x):
        /// shifts the number x to the right by 255 bits:
        /// IF the number is negative, the leftmost bit (bit 255) will be 1
        /// IF the number is positive,the leftmost bit (bit 255) will be 0

        /// sub(0, shr(255, x)):
        /// creates a mask of all 1s if x is negative
        /// creates a mask of all 0s if x is positive
        let mask := sub(0, shr(255, y))

        /// If x is negative, this effectively negates the number
        // if x is positive, it leaves the number unchanged, thereby computing the absolute value
        z := xor(mask, add(mask, y))

        /// shr(128, z):
        /// shifts the number z to the right by 128 bits after computing the absolute value
        z := shr(128, z)
    }
}
```

Recommendation:

As a potential optimal alternative, we advise the C-level assembly solution described **here** to be utilized instead. In its Solidity implementation, it incurs less gas than the current `MathLib::abs128` implementation while producing identical results.

Pollution of the bit space of the `x` variable is of no concern as long as inputs to the `MathLib::abs128` are cast safely.

Alleviation:

The Kwenta team reproduced the referenced C snippet in the `assembly` code of the referenced block, greatly optimizing the gas cost of assessing a 128-bit signed variable's absolute value.

Finding Types

A description of each finding type included in the report can be found below and is linked by each respective finding. A full list of finding types Omniscia has defined will be viewable at the central audit methodology we will publish soon.

Input Sanitization

As there are no inherent guarantees to the inputs a function accepts, a set of guards should always be in place to sanitize the values passed in to a particular function.

Indeterminate Code

These types of issues arise when a linked code segment may not behave as expected, either due to mistyped code, convoluted if blocks, overlapping functions / variable names and other ambiguous statements.

Language Specific

Language specific issues arise from certain peculiarities that the Circom language boasts that discerns it from other conventional programming languages.

Curve Specific

Circom defaults to using the BN128 scalar field (a 254-bit prime field), but it also supports BSL12-381 (which has a 255-bit scalar field) and Goldilocks (with a 64-bit scalar field). However, since there are no constants denoting either the prime or the prime size in bits available in the Circom language, some Circomlib templates like `Sign` (which returns the sign of the input signal), and `AliasCheck` (used by the strict versions of `Num2Bits` and `Bits2Num`), hardcode either the BN128 prime size or some other constant related to BN128. Using these circuits with a custom prime may thus lead to unexpected results and should be avoided.

Code Style

In these types of findings, we identify whether a project conforms to a particular naming convention and whether that convention is consistent within the codebase and legible. In case of inconsistencies, we point them out under this category. Additionally, variable shadowing falls under this category as well which is identified when a local-level variable contains the same name as a toplevel variable in the circuit.

Mathematical Operations

This category is used when a mathematical issue is identified. This implies an issue with the implementation of a calculation compared to the specifications.

Logical Fault

This category is a bit broad and is meant to cover implementations that contain flaws in the way they are implemented, either due to unimplemented functionality, unaccounted-for edge cases or similar extraordinary scenarios.

Privacy Concern

This category is used when information that is meant to be kept private is made public in some way.

Proof Concern

Under-constrained signals are one of the most common issues in zero-knowledge circuits. Issues with proof generation fall under this category.

Disclaimer

The following disclaimer applies to all versions of the audit report produced (preliminary / public / private) and is in effect for all past, current, and future audit reports that are produced and hosted under Omniscia:

IMPORTANT TERMS & CONDITIONS REGARDING OUR SECURITY AUDITS/REVIEWS/REPORTS AND ALL PUBLIC/PRIVATE CONTENT/DELIVERABLES

Omniscia ("Omniscia") has conducted an independent security review to verify the integrity of and highlight any vulnerabilities, bugs or errors, intentional or unintentional, that may be present in the codebase that were provided for the scope of this Engagement.

Blockchain technology and the cryptographic assets it supports are nascent technologies. This makes them extremely volatile assets. Any assessment report obtained on such volatile and nascent assets may include unpredictable results which may lead to positive or negative outcomes.

In some cases, services provided may be reliant on a variety of third parties. This security review does not constitute endorsement, agreement or acceptance for the Project and technology that was reviewed. Users relying on this security review should not consider this as having any merit for financial advice or technological due diligence in any shape, form or nature.

The veracity and accuracy of the findings presented in this report relate solely to the proficiency, competence, aptitude and discretion of our auditors. Omniscia and its employees make no guarantees, nor assurance that the contracts are free of exploits, bugs, vulnerabilities, deprecation of technologies or any system / economical / mathematical malfunction.

This audit report shall not be printed, saved, disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Omniscia.

All the information/opinions/suggestions provided in this report does not constitute financial or investment advice, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report.

Information in this report is provided 'as is'. Omniscia is under no covenant to the completeness, accuracy or solidity of the contracts reviewed. Omniscia's goal is to help reduce the attack vectors/surface and the high level of variance associated with utilizing new and consistently changing technologies.

Omniscia in no way claims any guarantee, warranty or assurance of security or functionality of the technology that was in scope for this security review.

In no event will Omniscia, its partners, employees, agents or any parties related to the design/creation of this security review be ever liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this security review.

Cryptocurrencies and all other technologies directly or indirectly related to cryptocurrencies are not standardized, highly prone to malfunction and extremely speculative by nature. No due diligence and/or safeguards may be insufficient and users should exercise maximum caution when participating and/or investing in this nascent industry.

The preparation of this security review has made all reasonable attempts to provide clear and actionable recommendations to the Project team (the “client”) with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts in scope for this engagement.

It is the sole responsibility of the Project team to provide adequate levels of test and perform the necessary checks to ensure that the contracts are functioning as intended, and more specifically to ensure that the functions contained within the contracts in scope have the desired intended effects, functionalities and outcomes, as documented by the Project team.

All services, the security reports, discussions, work product, attack vectors description or any other materials, products or results of this security review engagement is provided "as is" and "as available" and with all faults, uncertainty and defects without warranty or guarantee of any kind.

Omniscia will assume no liability or responsibility for delays, errors, mistakes, or any inaccuracies of content, suggestions, materials or for any loss, delay, damage of any kind which arose as a result of this engagement/security review.

Omniscia will assume no liability or responsibility for any personal injury, property damage, of any kind whatsoever that resulted in this engagement and the customer having access to or use of the products, engineers, services, security report, or any other other materials.

For avoidance of doubt, this report, its content, access, and/or usage thereof, including any associated services or materials, shall not be considered or relied upon as any form of financial, investment, tax, legal, regulatory, or any other type of advice.

Finding Types

A description of each finding type included in the report can be found below and is linked by each respective finding. A full list of finding types Omniscia has defined will be viewable at the central audit methodology we will publish soon.

Input Sanitization

As there are no inherent guarantees to the inputs a function accepts, a set of guards should always be in place to sanitize the values passed in to a particular function.

Indeterminate Code

These types of issues arise when a linked code segment may not behave as expected, either due to mistyped code, convoluted if blocks, overlapping functions / variable names and other ambiguous statements.

Language Specific

Language specific issues arise from certain peculiarities that the Circom language boasts that discerns it from other conventional programming languages.

Curve Specific

Circom defaults to using the BN128 scalar field (a 254-bit prime field), but it also supports BSL12-381 (which has a 255-bit scalar field) and Goldilocks (with a 64-bit scalar field). However, since there are no constants denoting either the prime or the prime size in bits available in the Circom language, some Circomlib templates like `Sign` (which returns the sign of the input signal), and `AliasCheck` (used by the strict versions of `Num2Bits` and `Bits2Num`), hardcode either the BN128 prime size or some other constant related to BN128. Using these circuits with a custom prime may thus lead to unexpected results and should be avoided.

Code Style

In these types of findings, we identify whether a project conforms to a particular naming convention and whether that convention is consistent within the codebase and legible. In case of inconsistencies, we point them out under this category. Additionally, variable shadowing falls under this category as well which is identified when a local-level variable contains the same name as a toplevel variable in the circuit.

Mathematical Operations

This category is used when a mathematical issue is identified. This implies an issue with the implementation of a calculation compared to the specifications.

Logical Fault

This category is a bit broad and is meant to cover implementations that contain flaws in the way they are implemented, either due to unimplemented functionality, unaccounted-for edge cases or similar extraordinary scenarios.

Privacy Concern

This category is used when information that is meant to be kept private is made public in some way.

Proof Concern

Under-constrained signals are one of the most common issues in zero-knowledge circuits. Issues with proof generation fall under this category.

Disclaimer

The following disclaimer applies to all versions of the audit report produced (preliminary / public / private) and is in effect for all past, current, and future audit reports that are produced and hosted under Omniscia:

IMPORTANT TERMS & CONDITIONS REGARDING OUR SECURITY AUDITS/REVIEWS/REPORTS AND ALL PUBLIC/PRIVATE CONTENT/DELIVERABLES

Omniscia ("Omniscia") has conducted an independent security review to verify the integrity of and highlight any vulnerabilities, bugs or errors, intentional or unintentional, that may be present in the codebase that were provided for the scope of this Engagement.

Blockchain technology and the cryptographic assets it supports are nascent technologies. This makes them extremely volatile assets. Any assessment report obtained on such volatile and nascent assets may include unpredictable results which may lead to positive or negative outcomes.

In some cases, services provided may be reliant on a variety of third parties. This security review does not constitute endorsement, agreement or acceptance for the Project and technology that was reviewed. Users relying on this security review should not consider this as having any merit for financial advice or technological due diligence in any shape, form or nature.

The veracity and accuracy of the findings presented in this report relate solely to the proficiency, competence, aptitude and discretion of our auditors. Omniscia and its employees make no guarantees, nor assurance that the contracts are free of exploits, bugs, vulnerabilities, deprecation of technologies or any system / economical / mathematical malfunction.

This audit report shall not be printed, saved, disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Omniscia.

All the information/opinions/suggestions provided in this report does not constitute financial or investment advice, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report.

Information in this report is provided 'as is'. Omniscia is under no covenant to the completeness, accuracy or solidity of the contracts reviewed. Omniscia's goal is to help reduce the attack vectors/surface and the high level of variance associated with utilizing new and consistently changing technologies.

Omniscia in no way claims any guarantee, warranty or assurance of security or functionality of the technology that was in scope for this security review.

In no event will Omniscia, its partners, employees, agents or any parties related to the design/creation of this security review be ever liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this security review.

Cryptocurrencies and all other technologies directly or indirectly related to cryptocurrencies are not standardized, highly prone to malfunction and extremely speculative by nature. No due diligence and/or safeguards may be insufficient and users should exercise maximum caution when participating and/or investing in this nascent industry.

The preparation of this security review has made all reasonable attempts to provide clear and actionable recommendations to the Project team (the “client”) with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts in scope for this engagement.

It is the sole responsibility of the Project team to provide adequate levels of test and perform the necessary checks to ensure that the contracts are functioning as intended, and more specifically to ensure that the functions contained within the contracts in scope have the desired intended effects, functionalities and outcomes, as documented by the Project team.

All services, the security reports, discussions, work product, attack vectors description or any other materials, products or results of this security review engagement is provided "as is" and "as available" and with all faults, uncertainty and defects without warranty or guarantee of any kind.

Omniscia will assume no liability or responsibility for delays, errors, mistakes, or any inaccuracies of content, suggestions, materials or for any loss, delay, damage of any kind which arose as a result of this engagement/security review.

Omniscia will assume no liability or responsibility for any personal injury, property damage, of any kind whatsoever that resulted in this engagement and the customer having access to or use of the products, engineers, services, security report, or any other other materials.

For avoidance of doubt, this report, its content, access, and/or usage thereof, including any associated services or materials, shall not be considered or relied upon as any form of financial, investment, tax, legal, regulatory, or any other type of advice.