OMNISCIA

June 24, 2025

SMART CONTRACT AUDIT REPORT

Kyber Network
Uniswap V4 Hooks

omniscia.io

info@omniscia.io

Online report: kyber-network-uniswap-v4-hooks

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

Online report: kyber-network-uniswap-v4-hooks

Uniswap V4 Hooks Security Audit

Audit Report Revisions

Commit Hash	Date	Audit Report Hash
da9dbb8984	May 6th 2025	f725f50dfd
323bfbb2bd	June 24th 2025	9aa9175044

Audit Overview

We were tasked with performing an audit of the Kyber Network codebase and in particular their novel Uniswap V4 Hook.

The Uniswap V4 hook implementation is meant to introduce the following features to the system:

- Restriction of swaps to a list of whitelisted addresses (i.e. the Kyberswap DEX Aggregator)
- Capture of excess outputs in swaps that would exceed the "fair market rate" of a particular pair

The system will calculate the "fair market rate" utilizing a signature validation mechanism, thereby indicating that this rate will be established utilizing off-chain means.

The capture of excess outputs (termed "Equilibrium Gains) occurs through an `IHooks::afterSwap` hook denoting the gain to be captured in the unspecified token currency.

As the system supports a one-way exchange rate mechanism, the `IHooks::beforeSwap` implementation restricts all swaps to represent exact-input swaps, ensuring fees can be appropriately imposed.

Over the course of the audit, we identified that the signature validation mechanism will not validate the sender that consumes the signature nor whether a particular signature has been consumed.

While it can be argued that the sender does not need to be validated, each signature should be consumed as it is meant to impose a `maxAmountIn` limitation that would be considered ineffective through signature re-use.

We advise the Kyber Network 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 Kyber Network 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 Kyber Network and have confirmed that all exhibits have been either adequately addressed or safely acknowledged.

We consider all outputs of the audit report properly consumed by the Kyber Network team with no outstanding remediative actions remaining.

Post-Audit Conclusion (Cont. I)

We evaluated the PancakeSwap KEM implementation and observed that all applicable findings of its Uniswap counterpart have been addressed.

During our analysis, we were able to identify a flaw in the way signature digests are generated.

We advise the Kyber Network team to inspect and promptly alleviate the relevant medium-severity exhibit of the audit report.

Post-Audit Conclusion (Cont. II)

The Kyber Network team evaluated the newly introduced exhibit and substantiated the fact that they do not believe it to be actively exploitable.

After validating that the issue is not applicable due to a difference in the underlying data structures between the Uniswap and PancakeSwap projects, we proceeded with lowering the exhibit's severity to minor.

The Kyber Network team opted to acknowledge the best-practice recommendation of distinguishing digests via project-specific prefixes, rendering all outputs of the audit report properly consumed by the Kyber Network team.

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	0	0	0	0
Informational	4	2	0	2
Minor	2	1	0	1
Medium	1	1	0	0
Major	0	0	0	0

During the audit, we filtered and validated a total of **1 findings utilizing static analysis** tools as well as identified a total of **6 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.

- **Scope**
- **Compilation**
- **Static Analysis**
- **Manual Review**
- **Code Style**

Scope

The audit engagement encompassed a specific list of contracts that were present in the commit hash of the repository that was in scope. The tables below detail certain meta-data about the target of the security assessment and a navigation chart is present at the end that links to the relevant findings per file.

Target

- Repository: **https://github.com/KyberNetwork/kyber-exclusive-amm-sc**
- Commit: da9dbb89845eb7f8da7ddc3beccd34d6cd48b446
- Language: Solidity
- Network: Ethereum, BNB Chain, Arbitrum, Optimism, Unichain, Base, Polygon, Avalanche
- Revisions: **da9dbb8984**, **323bfbb2bd**

Contracts Assessed

File	Total Finding(s)
src/base/BaseKEMHook.sol (BKE)	1
src/libraries/HookDataDecoder.sol (HDD)	1
src/base/Rescuable.sol (REL)	1
src/UniswapV4KEMHook.sol (UVK)	3
src/PancakeSwapInfinityKEMHook.sol (PVK)	1

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 `0.8.26` based on the version specified within the `foundry.toml` file.

The project contains discrepancies with regards to the Solidity version used, however, they are solely contained in external dependencies and can thus be safely ignored.

The `pragma` statements are locked to `0.8.26` (`=0.8.26`), 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 **14 potential issues** within the codebase of which **13 were ruled out to be false positives** or negligible findings.

The remaining **1 issues** were validated and grouped and formalized into the **1 exhibits** that follow:

ID	Severity	Addressed	Title
UVK-01S	Informational	Acknowledged	Inexistent Sanitization of Input Address

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in Kyber Network's Uniswap V4 hook contract.

As the project at hand implements a novel Uniswap V4 hook system, intricate care was put into ensuring that the **deltas reported by the system conform to the specifications and restrictions** laid forth within the Uniswap V4 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 signature-related vulnerabilities** within the system which could have had **minor-to-moderate ramifications** to its overall operation; for more information, kindly consult the relevant non-informational exhibits within the audit report as well as its summary.

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 descriptive documentation outlining the purpose of the system in detail.

A total of **6 findings** were identified over the course of the manual review of which **4 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 below enumerates all these security / behavioural findings:

ID	Severity	Addressed	Title
BKE-01M	Informational	Acknowledged	Inexistent Restriction of Status
PVK-01M	Minor	Acknowledged	Overlap of Signature Digests
UVK-01M	Minor	Yes	Inexistent Restriction of Signature Submitter
UVK-02M	Medium	Yes	Inexistent Consumption of Signature

Code Style

During the manual portion of the audit, we identified **2 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
HDD-01C	Informational	Yes	Incorrect Documentation
REL-01C	Informational	Yes	Non-Standard Library Usage

UniswapV4KEMHook Static Analysis Findings

UVK-01S: Inexistent Sanitization of Input Address

Type	Severity	Location
Input Sanitization	Informational	UniswapV4KEMHook.sol:L32-L50

Description:

The linked function accepts an `address` argument yet does not properly sanitize it.

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/UniswapV4KEMHook.sol

SOL

```sol
constructor(
  IPoolManager _poolManager,
  address initialOwner,
  address[] memory initialClaimableAccounts,
  address[] memory initialWhitelistedAccounts,
  address initialQuoteSigner,
  address initialEgRecipient
)
  BaseKEMHook(
    initialOwner,
```

Example (Cont.):

```sol
    initialClaimableAccounts,
    initialWhitelistedAccounts,
    initialQuoteSigner,
    initialEgRecipient
  )
{
  poolManager = _poolManager;
  Hooks.validateHookPermissions(IHooks(address(this)), getHookPermissions());
}
```

Recommendation:

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

Alleviation:

The Kyber Network team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

BaseKEMHook Manual Review Findings

BKE-01M: Inexistent Restriction of Status

Type	Severity	Location
Input Sanitization	Informational	**BaseKEMHook.sol**: • I-1: **L59-L65** • I-2: **L67-L73**

Description:

The `BaseKEMHook::_updateClaimable` and `BaseKEMHook::_updateWhitelisted` functions may emit the same `UpdateClaimable` / `UpdateWhitelisted` event multiple times for the same address and status combination as the `newStatus` is not actually validated as being a new state.

Impact:

As this exhibit solely impacts event emissions, its severity is capped to informational.

Example:

src/base/BaseKEMHook.sol

SOL

```sol
function _updateClaimable(address[] memory accounts, bool newStatus) internal {
  for (uint256 i = 0; i < accounts.length; i++) {
    claimable[accounts[i]] = newStatus;

    emit UpdateClaimable(accounts[i], newStatus);
  }
}

function _updateWhitelisted(address[] memory accounts, bool newStatus) internal {
  for (uint256 i = 0; i < accounts.length; i++) {
```

Example (Cont.):

```sol
69      whitelisted[accounts[i]] = newStatus;
70
71      emit UpdateWhitelisted(accounts[i], newStatus);
72    }
73  }
```

Recommendation:

We advise the code to ensure that the claimable / whitelisted state of an address actually changes by each respective function, preventing misleading event emissions.

Alleviation:

The Kyber Network team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

PancakeSwapInfinityKEMHook Manual Review Findings

PVK-01M: Overlap of Signature Digests

Type	Severity	Location
Logical Fault	Minor	PancakeSwapInfinityKEMHook.sol:L124-L135

Description:

The PancakeSwap variant of the Uniswap KEM implementation re-uses the same signature payload during swap operations.

As the signature payload hashes do not utilize contextual variables such as the contract address and are not prefixed with a domain separator, the same signature can be reused across implementations.

Impact:

Signature payloads for the same swap configuration can be reused at least once for the same nonce across implementations.

Example:

src/PancakeSwapInfinityKEMHook.sol

SOL

```sol
124 bytes32 digest = keccak256(
125   abi.encode(
126     sender,
127     key,
128     params.zeroForOne,
129     maxAmountIn,
130     maxExchangeRate,
131     exchangeRateDenom,
132     nonce,
133     expiryTime
```

Example (Cont.):

```sol
  )
);
require(
  SignatureChecker.isValidSignatureNow(quoteSigner, digest, signature),
InvalidSignature()
);
```

Recommendation:

We advise the signature payload to be prefixed with a unique identifier per KEM implementation, preventing the behaviour specified.

Alleviation:

The Kyber Network team evaluated this exhibit and opted to acknowledge it as they do not consider it to be a vulnerability due to the difference in the `PoolKey` structures between the Uniswap and PancakeSwap projects.

Specifically, the PancakeSwap data structure contains a single extraneous variable rendering digests to be unique for the same configurations across the two projects.

Nevertheless, we advised the PancakeSwap team to implement proper domains for each project which they opted to acknowledge due to the aforementioned difference between the data types. As such, we consider this exhibit safely acknowledged.

UniswapV4KEMHook Manual Review Findings

UVK-01M: Inexistent Restriction of Signature Submitter

Type	Severity	Location
Logical Fault	Minor	UniswapV4KEMHook.sol:L124-L135

Description:

The signature validation process in the `UniswapV4KEMHook::beforeSwap` function will not incorporate the whitelisted `sender` into the signed payload, permitting any of the multiple whitelisted senders to utilize a single signature.

Impact:

Any `whitelisted` member can presently utilize any signature as long as it is valid which we consider incorrect.

If signatures were being consumed, this exhibit would have been considered to be of medium severity, however, the present flaw simply permits a higher degree of access than expected.

Example:

src/UniswapV4KEMHook.sol

SOL

```sol
102 function beforeSwap(
103   address sender,
104   PoolKey calldata key,
105   IPoolManager.SwapParams calldata params,
106   bytes calldata hookData
107 ) external view onlyPoolManager returns (bytes4, BeforeSwapDelta, uint24) {
108   require(whitelisted[sender], NonWhitelistedAccount(sender));
109   require(params.amountSpecified < 0, ExactOutputDisabled());
110
111   (
```

Example (Cont.):

```sol
    int256 maxAmountIn,
    int256 maxExchangeRate,
    int256 exchangeRateDenom,
    uint256 expiryTime,
    bytes memory signature
  ) = HookDataDecoder.decodeAllHookData(hookData);

  require(block.timestamp <= expiryTime, ExpiredSignature(expiryTime, block.timestamp));
  require(
    -params.amountSpecified <= maxAmountIn,
    ExceededMaxAmountIn(maxAmountIn, -params.amountSpecified)
  );
  require(
    SignatureChecker.isValidSignatureNow(
      quoteSigner,
      keccak256(
        abi.encode(
          key, params.zeroForOne, maxAmountIn, maxExchangeRate, exchangeRateDenom, expiryTime
        )
      ),
      signature
    ),
    InvalidSignature()
  );

  return (this.beforeSwap.selector, BeforeSwapDeltaLibrary.ZERO_DELTA, 0);
}
```

Recommendation:

We advise the `sender` to be incorporated into the signed payload so as to prevent the consumption of signatures meant for different senders.

Alleviation:

The `sender` has been incorporated within the signature payload itself, rendering the original `whitelisted` enforcement redundant and alleviating the vulnerability described.

UVK-02M: Inexistent Consumption of Signature

Type	Severity	Location
Logical Fault	Medium	UniswapV4KEMHook.sol:L124-L135

Description:

The `quoteSigner` signature can be replayed indefinitely before its `expiryTime` as it is not consumed.

Impact:

A signed payload can be repeated an infinite number of times, effectively bypassing the `maxAmountIn` restriction.

Example:

src/UniswapV4KEMHook.sol

SOL

```sol
124 require(
125   SignatureChecker.isValidSignatureNow(
126     quoteSigner,
127     keccak256(
128       abi.encode(
129         key, params.zeroForOne, maxAmountIn, maxExchangeRate, exchangeRateDenom,
expiryTime
130       )
131     ),
132     signature
133   ),
```

Example (Cont.):

```sol
  InvalidSignature()
);

return (this.beforeSwap.selector, BeforeSwapDeltaLibrary.ZERO_DELTA, 0);
```

Recommendation:

We advise the signature to be marked as consumed by utilizing a nonce per `quoteSigner`, preventing signature replay attacks.

Alleviation:

An `UnorderedNonce` implementation has been introduced to the codebase that permits nonces to be marked as consumed, allowing nonces to be incorporated to the signed payload consumed for each **`UniswapV4KEMHook::beforeSwap`** operation.

HookDataDecoder Code Style Findings

HDD-01C: Incorrect Documentation

Type	Severity	Location
Code Style	Informational	HookDataDecoder.sol:L34

Description:

The referenced documentation does not match the decoding approach utilized by the `HookDataDecoder::decodeExchangeRate` function.

Example:

src/libraries/HookDataDecoder.sol

SOL

```sol
/// @dev equivalent to: abi.decode(params, (int256, int256, int256, uint256, bytes)) in calldata
function decodeExchangeRate(bytes calldata hookData)
  internal
  pure
  returns (int256 maxExchangeRate, int256 exchangeRateDenom)
{
  assembly ("memory-safe") {
    maxExchangeRate := calldataload(add(hookData.offset, 0x20))
    exchangeRateDenom := calldataload(add(hookData.offset, 0x40))
  }
```

Example (Cont.):

```sol
}
```

Recommendation:

We advise it to be corrected, increasing the code's legibility.

Alleviation:

The documentation has been corrected as advised, addressing this exhibit.

Rescuable Code Style Findings

REL-01C: Non-Standard Library Usage

Type	Severity	Location
Code Style	Informational	Rescuable.sol:L12, L86

Description:

The `SafeERC20` library is being applied to the `IERC20` data type yet is invoked directly (i.e. `SafeERC20.safeTransfer`) rather than through the data type it is applied to (i.e. `token.safeTransfer`).

Example:

src/base/Rescuable.sol

```sol
86  SafeERC20.safeTransfer(token, recipient, amount);
```

Recommendation:

We advise the library's usage to be streamlined, optimizing the code's legibility.

Alleviation:

The library's usage style has been streamlined, addressing this exhibit.

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.

Severity Definition

In the ever-evolving world of blockchain technology, vulnerabilities continue to take on new forms and arise as more innovative projects manifest, new blockchain-level features are introduced, and novel layer-2 solutions are launched. When performing security reviews, we are tasked with classifying the various types of vulnerabilities we identify into subcategories to better aid our readers in understanding their impact.

Within this page, we will clarify what each severity level stands for and our approach in categorizing the findings we pinpoint in our audits. To note, all severity assessments are performed **as if the contract's logic cannot be upgraded** regardless of the underlying implementation.

Severity Levels

There are five distinct severity levels within our reports; `unknown`, `informational`, `minor`, `medium`, and `major`. A TL;DR overview table can be found below as well as a dedicated chapter to each severity level:

	Impact (None)	Impact (Low)	Impact (Moderate)	Impact (High)
Likelihood (None)	Informational	Informational	Informational	Informational
Likelihood (Low)	Informational	Minor	Minor	Medium
Likelihood (Moderate)	Informational	Minor	Medium	Major
Likelihood (High)	Informational	Medium	Major	Major

Unknown Severity

The `unknown` severity level is reserved for misbehaviors we observe in the codebase that cannot be quantified using the above metrics. Examples of such vulnerabilities include potentially desirable system behavior that is undocumented, reliance on external dependencies that are out-of-scope but could result in some form of vulnerability arising, use of external out-of-scope contracts that appears incorrect but cannot be pinpointed, and other such vulnerabilities.

In general, `unknown` severity level vulnerabilities require follow-up information by the project being audited and are either adjusted in severity (if valid), or marked as nullified (if invalid).

Additionally, the `unknown` severity level is sometimes assigned to centralization issues that cannot be assessed in likelihood due to their exploitation being tied to the honesty of the project's team.

Informational Severity

The `informational` severity level is dedicated to findings that do not affect the code functionally and tend to be stylistic or optimizational in nature. Certain edge cases are also set under `informational` vulnerabilities, such as overflow operations that will not manifest in the lifetime of the contract but should be guarded against as a best practice, to give an example.

Minor Severity

The `minor` severity level is meant for vulnerabilities that require functional changes in the code but tend to either have little impact or be unlikely to be recreated in a production environment. These findings can be acknowledged except for findings with a moderate impact but low likelihood which must be alleviated.

Medium Severity

The `medium` severity level is assigned to vulnerabilities that must be alleviated and have an observable impact on the overall project. These findings can only be acknowdged if the project deems them desirable behavior and we disagree with their point-of-view, instead urging them to reconsider their stance while marking the exhibit as acknowledged given that the project has ultimate say as to what vulnerabilities they end up patching in their system.

Major Severity

The `major` severity level is the maximum that can be specified for a finding and indicates a significant flaw in the code that must be alleviated.

Likelihood & Impact Assessment

As the preface chapter specifies, the blockchain space is constantly reinventing itself meaning that new vulnerabilities take place and our understanding of what security means differs year-to-year.

In order to reliably assess the likelihood and impact of a particular vulnerability, we instead apply an abstract measurement of a vulnerability's impact, duration the impact is applied for, and probability that the vulnerability would be exploited in a production environment.

Our proposed definitions are inspired by multiple sources in the security community and are as follows:

- Impact (High): A core invariant of the protocol can be broken for an extended duration.
- Impact (Moderate): A non-core invariant of the protocol can be broken for an extended duration or at scale, or an otherwise major-severity issue is reduced due to hypotheticals or external factors affecting likelihood.
- Impact (Low): A non-core invariant of the protocol can be broken with reduced likelihood or impact.
- Impact (None): A code or documentation flaw whose impact does not achieve low severity, or an issue without theoretical impact; a valuable best-practice
- Likelihood (High): A flaw in the code that can be exploited trivially and is ever-present.
- Likelihood (Moderate): A flaw in the code that requires some external factors to be exploited that are likely to manifest in practice.
- Likelihood (Low): A flaw in the code that requires multiple external factors to be exploited that may manifest in practice but would be unlikely to do so.
- Likelihood (None): A flaw in the code that requires external factors proven to be impossible in a production environment, either due to mathematical constraints, operational constraints, or system-related factors (i.e. EIP-20 tokens not being re-entrant).

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.