OMNISCIA

September 2, 2024

SMART CONTRACT AUDIT REPORT

Fermion Protocol
Second Round

omniscia.io

info@omniscia.io

Online report: fermion-protocol-second-round

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: fermion-protocol-second-round

Second Round Security Audit

Audit Report Revisions

Commit Hash	Date	Audit Report Hash
de82a06de3	August 27th 2024	8cb0dec386
57c39a4a03	September 2nd 2024	bf42b789f2

Audit Overview

We were tasked with performing an audit of the Fermion Protocol codebase and in particular a follow-up round on the codebase to cover **EIP-2771** meta-transaction support as well as a hotfix for a Denial-of-Service attack identified by the Fermion Protocol team.

The vulnerability identified by the Fermion Protocol team would be considered a griefing attack whereby a malicious user could directly interact with the Boson Protocol price discovery contract to cause either of the following:

So as to resolve this vulnerability, a new `Unwrapping` state was introduced that is toggled prior to the interaction with the price discovery contract thereby preventing the aforementioned vulnerability.

In addition to the hotfix for the above issue, the code was slightly refactored to properly support **EIP-2771** meta-transactions at the `FermionFNFT` level via the Fermion Protocol diamond and naming conventions around account roles (previously called wallets) were updated.

Over the course of the audit, we identified two latent vulnerabilities that could manifest in the future as a result of the special **EIP-2771** meta-transaction integration between the Fermion Protocol Diamond and the `FermionFNFT` instance as well as improper backward compatibility by the `MetaTransactionFacet`.

We advise the Fermion Protocol 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 Fermion Protocol 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 Fermion Protocol and have identified that all exhibits have been adequately dealt with no outstanding issues remaining in the report.

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	0	0	0	0
Informational	8	8	0	0
Minor	1	1	0	0
Medium	0	0	0	0
Major	0	0	0	0

During the audit, we filtered and validated a total of **0 findings utilizing static analysis** tools as well as identified a total of **9 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

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/fermionprotocol/contracts**
- Commit: de82a06de351ba09e76d40e10dfe0ddbdbac269d
- Language: Solidity
- Network: Ethereum, Polygon
- Revisions: **de82a06de3**, **57c39a4a03**

Contracts Assessed

File	Total Finding(s)
contracts/protocol/libs/Access.sol (ASS)	0
contracts/diamond/facets/AccessController.sol (ACR)	0
contracts/protocol/clients/Common.sol (CNO)	0
contracts/protocol/facets/Config.sol (CGI)	0
contracts/protocol/libs/Context.sol (CTX)	0
contracts/protocol/facets/Custody.sol (CYD)	0
contracts/protocol/domain/Constants.sol (CST)	0
contracts/protocol/libs/CustodyLib.sol (CLB)	0
contracts/protocol/facets/CustodyVault.sol (CVT)	0
contracts/diamond/Diamond.sol (DDN)	0

contracts/diamond/facets/DiamondCutFacet.sol (DCF)	0
contracts/diamond/facets/DiamondLoupeFacet.sol (DLF)	0
contracts/protocol/facets/Entity.sol (EYT)	0
contracts/protocol/domain/Errors.sol (ESR)	0
contracts/protocol/libs/EntityLib.sol (ELB)	0
contracts/protocol/facets/Funds.sol (FSD)	0
contracts/protocol/libs/FundsLib.sol (FLB)	1
contracts/protocol/clients/FermionFNFT.sol (FFN)	0
contracts/protocol/libs/FermionFNFTLib.sol (FFF)	3
contracts/protocol/clients/FermionWrapper.sol (FWR)	0
contracts/protocol/clients/FermionFNFTBase.sol (FFT)	0
contracts/protocol/clients/FermionFractions.sol (FFS)	0
contracts/protocol/clients/FermionFractionsERC20Base.sol (FFE)	0
contracts/protocol/facets/Initialization.sol (INO)	0

contracts/diamond/libraries/LibDiamond.sol (LDD)	0
contracts/protocol/facets/MetaTransaction.sol (MTN)	4
contracts/protocol/facets/Offer.sol (ORE)	1
contracts/protocol/facets/Pause.sol (PES)	0
contracts/protocol/libs/ReentrancyGuard.sol (RGD)	0
contracts/protocol/libs/Storage.sol (SEG)	0
contracts/protocol/clients/SeaportWrapper.sol (SWR)	0
contracts/protocol/domain/Types.sol (TSE)	0
contracts/protocol/facets/Verification.sol (VNO)	0

Compilation

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

To compile the project, the `compile` command needs to be issued via the `npx` CLI tool to `hardhat`:

```bash
npx hardhat compile
```

The `hardhat` tool automatically selects Solidity version `0.8.24` based on the version specified within the `hardhat.config.ts` 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 have been locked to `0.8.24` (`=0.8.24`), the same version utilized for our static analysis as well as optimizational review of the codebase.

During compilation with the `hardhat` 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 **114 potential issues** within the codebase of which **114 were ruled out to be false positives** or negligible findings.

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in Fermion Protocol's updated version.

As the project at hand implements EIP-2771 integration, intricate care was put into ensuring that the **authorization flow 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 a backward compatibility discrepancy** within the system which could have had **minor ramifications** to its overall operation as well as two latent vulnerabilities; for more information, kindly consult the relevant exhibits within the audit report as well as the report's 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 an exemplary extent, containing extensive in-line documentation that precisely describes the code's behavior.

A total of **9 findings** were identified over the course of the manual review of which **3 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
FLB-01M	Informational	Yes	Potentially Significant Latent Vulnerability
MTN-01M	Minor	Yes	Improper Backward Compatibility
ORE-01M	Informational	Yes	Potentially Insecure External Call

Code Style

During the manual portion of the audit, we identified **6 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
FFF-01C	Informational	Yes	Inefficient ABI Encoding Mechanisms
FFF-02C	Informational	Yes	Inexplicable Decoding Discrepancy
FFF-03C	Informational	Yes	Repetitive Value Literal
MTN-01C	Informational	Yes	Potential Optimization of Offer ID Management
MTN-02C	Informational	Yes	Redundant Type Cast
MTN-03C	Informational	Yes	Suboptimal Struct Declaration Style

FundsLib Manual Review Findings

FLB-01M: Potentially Significant Latent Vulnerability

Type	Severity	Location
Logical Fault	Informational	FundsLib.sol:L61-L104

Description:

The `FundsLib::transferFundsToProtocol` that can be invoked with arbitrary arguments via the `FundsFacet::depositFunds` may end up interacting with a `FermionFNFT` instance via the `IERC20::transferFrom` function signature which can result in misbehaviors based on the newly introduced **EIP-2771** meta-transaction support.

To note, the code is presently not vulnerable as a side-effect to an earlier recommendation of our team to prevent **EIP-721** assets from being transferred, however, it should be noted that the severity of an exploitation of this vector would be major.

Impact:

While presently secure, the security arises as a side-effect of an earlier remediation rather than an explicit safe-guard.

Example:

contracts/protocol/libs/FundsLib.sol

SOL

```sol
61  function transferFundsToProtocol(address _tokenAddress, address _from, uint256
_amount) internal {
62      // prevent ERC721 deposits
63      (bool success, bytes memory returnData) = _tokenAddress.staticcall(
64          abi.encodeCall(IERC165.supportsInterface, (type(IERC721).interfaceId))
65      );
66
67      if (success) {
68          if (returnData.length != 32) {
69              revert FermionGeneralErrors.UnexpectedDataReturned(returnData);
70          } else {
```

Example (Cont.):

```sol
            // If returned value equals 1 (= true), the contract is ERC721 and we should revert
            uint256 result = abi.decode(returnData, (uint256)); // decoding into uint256 not bool to cover all cases
            if (result == 1) {
                revert FundsErrors.ERC721NotAllowed(_tokenAddress);
            } else if (result > 1) {
                revert FermionGeneralErrors.UnexpectedDataReturned(returnData);
            }
            // If returned value equals 0 (= false), the contract is not ERC721 and we can continue.
        }
    } else {
        if (returnData.length == 0) {
            // Do nothing. ERC20 not implementing IERC721 interface is expected to revert without reason
        } else {
            // If an actual error message is returned, revert with it
            /// @solidity memory-safe-assembly
            assembly {
                revert(add(32, returnData), mload(returnData))
            }
        }
    }

    // protocol balance before the transfer
    uint256 protocolTokenBalanceBefore = IERC20(_tokenAddress).balanceOf(address(this));

    // transfer ERC20 tokens from the caller
    IERC20(_tokenAddress).safeTransferFrom(_from, address(this), _amount);

    // protocol balance after the transfer
```

Example (Cont.):

```sol
99      uint256 protocolTokenBalanceAfter =
IERC20(_tokenAddress).balanceOf(address(this));
100
101     // make sure that expected amount of tokens was transferred
102     uint256 receivedAmount = protocolTokenBalanceAfter -
protocolTokenBalanceBefore;
103     if (receivedAmount != _amount) revert FundsErrors.WrongValueReceived(_amount,
receivedAmount);
104 }
```

Recommendation:

We advise proper documentation to be introduced above the **EIP-721** interface compatibility check that clarifies the check is imperative to the secure operation of the Fermion Protocol so as to avoid interactions with `FermionFNFT` contracts.

Alleviation:

Adequate documentation was introduced to outline that interactions with the `FermionFNFT` contract should either be prohibited or handled uniquely to ensure the `_msgSender()` resolves to a securely-deduced address.

MetaTransaction Manual Review Findings

MTN-01M: Improper Backward Compatibility

Type	Severity	Location
Logical Fault	Minor	MetaTransaction.sol:L95

Description:

The `MetaTransactionFacet::executeMetaTransaction` function variant that is meant to be backward compatible has had its `payable` modifier removed, resulting in improper backward compatibility.

Impact:

The `MetaTransactionFacet::executeMetaTransaction` is presently not backward compatible when it comes to transactions concerning native payments despite what it claims.

Example:

contracts/protocol/facets/MetaTransaction.sol

SOL

```sol
function executeMetaTransaction(
    address _userAddress,
    string calldata _functionName,
    bytes calldata _functionSignature,
    uint256 _nonce,
    bytes32 _sigR,
    bytes32 _sigS,
    uint8 _sigV
) external {
    executeMetaTransaction(
```

Example (Cont.):

```sol
        address(this),
        _userAddress,
        _functionName,
        _functionSignature,
        _nonce,
        Signature(_sigR, _sigS, _sigV),
        0
    );
}
```

Recommendation:

We advise the `payable` keyword to be re-introduced to the function, ensuring that it properly acts as a backward compatible variant of the `MetaTransactionFacet::executeMetaTransaction` function.

Alleviation:

The `payable` function modifier was introduced as well as a `bytes memory` return argument ensuring backward compatibility is properly maintained.

Offer Manual Review Findings

ORE-01M: Potentially Insecure External Call

Type	Severity	Location
Standard Conformity	Informational	Offer.sol:L558-L564

Description:

The `OfferFacet::wrapNFTS` function will invoke the `IFermionFNFT::initialize` function which in turn invokes initializers in external OpenZeppelin dependencies.

After the **EIP-2771** related updates to the `FermionFNFT` contract, the `ERC2771Context::_msgSender` will evaluate to an incorrect value during an invocation of `FermionFNFT::initialize` as presently executed.

While this does not presently pose a security threat, any update of the upstream OpenZeppelin dependencies may result in a vulnerability manifesting.

Impact:

The recent **EIP-2771** related updates performed to the `FermionFNFT` contract present a latent vulnerability for the contract's initialization.

Example:

contracts/protocol/facets/Offer.sol

SOL

```
538 function wrapNFTS(
539     uint256 _offerId,
540     IBosonVoucher _bosonVoucher,
541     uint256 _startingNFTId,
542     uint256 _quantity,
543     FermionStorage.ProtocolStatus storage ps
544 ) internal {
545     address msgSender = _msgSender();
546     FermionStorage.OfferLookups storage offerLookup =
FermionStorage.protocolLookups().offerLookups[_offerId];
547
```

Example (Cont.):

```sol
548     address wrapperAddress = offerLookup.fermionFNFTAddress;
549     if (wrapperAddress == address(0)) {
550         // Currently, the wrapper is created for each offer, since
BOSON_PROTOCOL.reserveRange can be called only once
551         // so else path is not possible. This is here for future proofing.
552
553         // create wrapper
554         wrapperAddress = Clones.cloneDeterministic(ps.fermionFNFTBeaconProxy,
bytes32(_offerId));
555         offerLookup.fermionFNFTAddress = wrapperAddress;
556
557         FermionTypes.Offer storage offer =
FermionStorage.protocolEntities().offer[_offerId];
558         IFermionFNFT(wrapperAddress).initialize(
559             address(_bosonVoucher),
560             msgSender,
561             offer.exchangeToken,
562             _offerId,
563             offer.metadataURI
564         );
565     }
566
567     // wrap NFTs
568     _bosonVoucher.setApprovalForAll(wrapperAddress, true);
569     wrapperAddress.wrapForAuction(_startingNFTId, _quantity, msgSender);
570     _bosonVoucher.setApprovalForAll(wrapperAddress, false);
571
572     emit NFTsWrapped(_offerId, wrapperAddress, _startingNFTId, _quantity);
573 }
```

Recommendation:

We advise a proper mechanism via the `FermionFNFTLib` library to be introduced, permitting the initialization of a `FermionFNFT` contract to be performed securely.

Alleviation:

A proper mechanism was introduced to the `FermionFNFTLib` library permitting initialization to yield a correct `_msgSender()` thus alleviating this exhibit in full.

FermionFNFTLib Code Style Findings

FFF-01C: Inefficient ABI Encoding Mechanisms

Type	Severity	Location
Gas Optimization	Informational	FermionFNFTLib.sol:L31, L37, L46, L61

Description:

The referenced ABI encoding mechanisms will utilize a string literal to generate the ABI payload with thereby performing a redundant `keccak256` evaluation on each invocation.

Example:

contracts/protocol/libs/FermionFNFTLib.sol

SOL

```sol
29  function transferFrom(address _fnft, address from, address to, uint256 tokenId)
internal {
30      _fnft.functionCallWithAddress(
31          abi.encodeWithSignature("transferFrom(address,address,uint256)", from, to,
tokenId)
32      );
33  }
```

Recommendation:

We advise a proper function signature to be imported via a relevant interface (i.e. `IERC20.transferFrom.selector`) and the ABI `encodeCall` built-in function to be utilized throughout the contract so as to optimize its gas cost.

Alleviation:

While two of the referenced ABI encoding mechanisms have been optimized as advised, the `mintFractions` functions remain unoptimized.

Given that they share the same function name and would thus require a distinct `interface` to be introduced for each, we consider the remediation of the two former instances to be adequate for considering this exhibit addressed.

FFF-02C: Inexplicable Decoding Discrepancy

Type	Severity	Location
Standard Conformity	Informational	FermionFNFTLib.sol:L30-L32, L40-L41

Description:

The `FermionFNFTLib::transferFrom` and `FermionFNFTLib::transfer` functions are discrepant between them as the former will not decode or yield a return value whereas the latter will yield a `bool` variable.

Impact:

The return data of the `FermionFNFTLib::transfer` function remains unused in the current iteration of the codebase, rendering either of the proposed approaches as adequate.

Example:

contracts/protocol/libs/FermionFNFTLib.sol

SOL

```sol
29  function transferFrom(address _fnft, address from, address to, uint256 tokenId)
internal {
30      _fnft.functionCallWithAddress(
31          abi.encodeWithSignature("transferFrom(address,address,uint256)", from, to,
tokenId)
32      );
33  }
34
35  function transfer(address _fnft, address to, uint256 value) internal returns
(bool) {
36      bytes memory returndata = _fnft.functionCallWithAddress(
37          abi.encodeWithSignature("transfer(address,uint256)", to, value)
38      );
```

Example (Cont.):

```sol

        if (returndata.length != 32) revert FermionGeneralErrors.UnexpectedDataReturned(returndata);
        return abi.decode(returndata, (bool));
    }
```

Recommendation:

We advise consistency across the two functions to be enforced, either yielding a `bool` variable in both functions or neither.

Alleviation:

The discrepancy was resolved by streamlining the behavior of the two functions to not yield any return argument.

FFF-03C: Repetitive Value Literal

Type	Severity	Location
Code Style	Informational	FermionFNFTLib.sol:L40, L87

Description:

The linked value literal is repeated across the codebase multiple times.

Example:

contracts/protocol/libs/FermionFNFTLib.sol

SOL

```sol
40  if (returndata.length != 32) revert
FermionGeneralErrors.UnexpectedDataReturned(returndata);
```

Recommendation:

We advise it to be set to a `constant` variable instead, optimizing the legibility of the codebase.

In case the `constant` has already been declared, we advise it to be properly re-used across the code.

Alleviation:

The referenced value literal `32` has been properly relocated to a contract-level `constant` declaration labelled `SLOT_SIZE`, optimizing the code's legibility.

MetaTransaction Code Style Findings

MTN-01C: Potential Optimization of Offer ID Management

Type	Severity	Location
Gas Optimization	Informational	MetaTransaction.sol:L228-L233

Description:

The updated `MetaTransactionFacet` system will accept a target `_contractAddress` to use for the meta-transaction as well as an offer ID that would link to the relevant `FermionFNFT` instance.

The current design is inefficient, as the `_offerId` itself can be utilized to deduce whether a call to the Diamond itself is being performed or a call to a `FermionFNFT` contract.

Example:

contracts/protocol/facets/MetaTransaction.sol

SOL

```
228 if (
229     (_offerId == 0 && _contractAddress != address(this)) ||
230     (_offerId > 0 &&
231         FermionStorage.protocolLookups().offerLookups[_offerId].fermionFNFTAddress
!= _contractAddress)
232 ) revert InvalidContractAddress(_contractAddress);
```

Recommendation:

We advise this optimization to be applied, greatly reducing the gas cost involved in executing meta-transactions via the `MetaTransactionFacet`.

Alleviation:

The code was updated to properly configure the `contractAddress` based on the input `_offerId` value, optimizing the code's gas cost.

MTN-02C: Redundant Type Cast

Type	Severity	Location
Code Style	Informational	MetaTransaction.sol:L290

Description:

The `_contractAddress` member is cast to the `address` data type redundantly as it is already represented in that type.

Example:

contracts/protocol/facets/MetaTransaction.sol

SOL

```sol
279 function executeTx(
280     address _contractAddress,
281     address _userAddress,
282     string calldata _functionName,
283     bytes calldata _functionSignature,
284     uint256 _nonce
285 ) internal returns (bytes memory) {
286     // Store the nonce provided to avoid playback of the same tx
287     FermionStorage.metaTransaction().usedNonce[_userAddress][_nonce] = true;
288
```

Example (Cont.):

```sol
    // Invoke local function with an external call
    (bool success, bytes memory returnData) = address(_contractAddress).call{ value: msg.value }(
        abi.encodePacked(_functionSignature, _userAddress)
    );
```

Recommendation:

We advise the casting operation to be omitted, optimizing the legibility of the codebase.

Alleviation:

The redundant type casting operation has been safely omitted from the codebase.

MTN-03C: Suboptimal Struct Declaration Style

Type	Severity	Location
Code Style	Informational	MetaTransaction.sol:L102

Description:

The linked declaration style of a struct is using index-based argument initialization.

Example:

contracts/protocol/facets/MetaTransaction.sol

SOL

```
102 Signature(_sigR, _sigS, _sigV),
```

Recommendation:

We advise the key-value declaration format to be utilized instead, greatly increasing the legibility of the codebase.

Alleviation:

The key-value declaration style is now properly in use within the referenced `struct` declaration, addressing this exhibit in full.

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:

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.