OMNISCIA

January 31, 2023

SMART CONTRACT AUDIT REPORT

Boson Protocol
v2.2


omniscia.io


info@omniscia.io


Online report: boson-core-protocol-v2.2

Compilation

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

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 `0.8.9` based on the version specified within the `hardhat.config.js` file.

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

The `pragma` versions have been locked to `0.8.9` (`=0.8.9`), 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 **129 potential issues** within the codebase of which **0 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
BVR-01S	Unknown	Yes	Inexistent Conformity to the Checks-Effects-Interactions Pattern

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in Boson Protocol's latest premint update.

As the project at hand implements an intricate adjustment to an EIP-721 implementation, intricate care was put into ensuring that the **flow of assets within the system conforms to the specifications and restrictions** laid forth within the protocol's specification and that **existing functionality of Boson Protocol properly integrates with preminted vouchers**.

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 **identified a vulnerability arising from a function contrasting its previous behaviour** within the system which could have had **moderate ramifications** to external integrators.

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 is exemplary, containing thorough in-line documentation of all functions present in the system.

A total of **16 findings** were identified over the course of the manual review of which **10 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
BVR-01M	Unknown	Yes	Inexistent Sanitization of Range Validity
BVR-02M	Unknown	Yes	Inexistent Validation of Range Length
BVR-03M	Unknown	Yes	Weak Enforcement of `_rangeOfferIds` Ascending Order
BVR-04M	Minor	Yes	Potential Usability Enhancement of Preminted Vouchers
BVR-05M	Medium	Yes	Incorrect Invocation of Voucher Transfer Hook
DRH-01M	Medium	Acknowledged	Inexistent Escalation Incentive System
EHF-01M	Medium	Yes	Potential Fatal Failure of Voucher Redemption
OBE-01M	Minor	Yes	Inexistent Integration of Unlimited Offers
OBE-02M	Minor	Yes	Unsatisfied Function Pre-Conditions
PIF-01M	Major	Yes	Unprotected Facet w/ Delegate 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
BVR-01C	Informational	Yes	Duplication of Data Point
BVR-02C	Informational	Yes	Generic Typographic Mistake
BVR-03C	Informational	Yes	Optimization of Binary Search Algorithm
DHF-01C	Informational	Yes	Inefficient Application of Region Pause Status Enforcement
EHF-01C	Informational	Yes	Redundant Duplication of Pre-Conditions
OH1-01C	Informational	Yes	Duplicate Enforcement of Pause Region Status

BosonVoucher Static Analysis Findings

BVR-01S: Inexistent Conformity to the Checks-Effects-Interactions Pattern

Type	Severity	Location
Language Specific	Unknown	BosonVoucher.sol:L645-L649, L650

Description:

The `_beforeTransferHook` function will first invoke `commitToPreMintedOffer` prior to issuing a `delete` operation on the `_premintStatus` struct, permitting a re-entrancy attack to potentially occur within a consequent call of `commitToPreMintedOffer`.

Impact:

The Checks-Effects-Interactions (CEI) pattern should always be applied in all function implementations to ensure no latent re-entrancy attacks are present that can manifest in a future update of the protocol that i.e. introduces unsafe calls to `commitToPreMintedOffer`. This scenario is not unlikely as one can assume that it is protected by the `nonReentrant` modifier whilst contracts interacting with it would be susceptible to re-entrancies.

Example:

contracts/protocol/clients/voucher/BosonVoucher.sol

SOL

```
631 function _beforeTokenTransfer(
632     address _from,
633     address _to,
634     uint256 _tokenId
635 ) internal override {
636     // Update the buyer associated with the voucher in the protocol
637     // Only when transferring, not minting or burning
638     if (_from == owner()) {
639         if (_premintStatus.committable) {
640             // Set the preminted token as committed
641             _committed[_tokenId] = true;
```

```
642
643             // If this is a transfer of preminted token, treat it differently
644             address protocolDiamond = IClientExternalAddresses(BeaconClientLib._beacon
645             IBosonExchangeHandler(protocolDiamond).commitToPreMintedOffer(
646                 payable(_to),
647                 _premintStatus.offerId,
648                 _tokenId
649             );
650             delete _premintStatus;
651         } else {
652             // Already committed, treat as a normal transfer
653             onVoucherTransferred(_tokenId, payable(_to));
654         }
655     } else if (_from != address(0) && _to != address(0) && _from != _to) {
656         onVoucherTransferred(_tokenId, payable(_to));
657     }
658 }
```

Recommendation:

Although in the current implementation `commitToPreMintedOffer` is `nonReentrant` and the function solely invokes read-only functions of untrusted contracts, no security vulnerability arises from the mis-applied pattern. In any case, we advise the `delete` instruction to be relocated prior to the external calls as the `_premintStatus` is not utilized in between.

Alleviation:

The statements were re-ordered per our recommendation, rendering the code secure against potential mid-execution re-entrancies of a corrupt `_premintStatus` data entry.

BosonVoucher Manual Review Findings

BVR-01M: Inexistent Sanitization of Range Validity

Type	Severity	Location
Mathematical Operations	Unknown	BosonVoucher.sol:L168, L169

Description:

The `reserveRange` function does not explicitly validate that the range defined can be adequately computed (i.e. does not overflow via addition in `issueVoucher`).

Impact:

The issue does not pose an active threat, however, this validation should be enforced locally to ensure the security guarantees of the `BosonVoucher` contract are upheld self-sufficiently.

Example:

contracts/protocol/clients/voucher/BosonVoucher.sol

SOL

```sol
/**
 * @notice Initializes the voucher.
 * This function is callable only once.
 */
function initializeVoucher(
    uint256 _sellerId,
    address _newOwner,
    VoucherInitValues calldata voucherInitValues
) public initializer {
    string memory sellerId = Strings.toString(_sellerId);
    string memory voucherName = string(abi.encodePacked(VOUCHER_NAME, " ", sellerId));
    string memory voucherSymbol = string(abi.encodePacked(VOUCHER_SYMBOL, "_", sellerI

    __ERC721_init_unchained(voucherName, voucherSymbol);

    // we don't call init on ownable, but rather just set the ownership to correct own
    _transferOwnership(_newOwner);

```

```
90      _setContractURI(voucherInitValues.contractURI);
91
92      _setRoyaltyPercentage(voucherInitValues.royaltyPercentage);
93
94      emit VoucherInitialized(_sellerId, _royaltyPercentage, _contractURI);
95  }
96
97  /**
98   * @notice Issues a voucher to a buyer.
99   *
100  * Minted voucher supply is sent to the buyer.
101  * Caller must have PROTOCOL role.
102  *
103  * Reverts if:
104  * - Exchange id falls within a reserved range
105  *
106  * @param _exchangeId - the id of the exchange (corresponds to the ERC-721 token id)
107  * @param _buyer - the buyer address
108  */
109 function issueVoucher(uint256 _exchangeId, address _buyer) external override onlyRole(
110     // Get the exchange
111     (, Exchange memory exchange) = getBosonExchange(_exchangeId);
112
113     // See if the offer id is associated with a range
114     Range storage range = _rangeByOfferId[exchange.offerId];
115
116     // Revert if exchange id falls within a reserved range
117     uint256 rangeStart = range.start;
118     require(
119         (_exchangeId < rangeStart) || (_exchangeId >= rangeStart + range.length),
120         EXCHANGE_ID_IN_RESERVED_RANGE
121     );
122
123     // Mint the voucher, sending it to the buyer address
124     _mint(_buyer, _exchangeId);
125 }
126
127 /**
128  * @notice Burns a voucher.
129  *
130  * Caller must have PROTOCOL role.
131  *
132  * @param _exchangeId - the id of the exchange (corresponds to the ERC-721 token id)
133  */
134 function burnVoucher(uint256 _exchangeId) external override onlyRole(PROTOCOL) {
135     _burn(_exchangeId);
136 }
```

```
137
138 /**
139  * @notice Reserves a range of vouchers to be associated with an offer
140  *
141  * Must happen prior to calling preMint
142  * Caller must have PROTOCOL role.
143  *
144  * Reverts if:
145  * - Start id is not greater than zero
146  * - Offer id is already associated with a range
147  *
148  * @param _offerId - the id of the offer
149  * @param _start - the first id of the token range
150  * @param _length - the length of the range
151  */
152 function reserveRange(
153     uint256 _offerId,
154     uint256 _start,
155     uint256 _length
156 ) external onlyRole(PROTOCOL) {
157     // Make sure range start is valid
158     require(_start > 0, INVALID_RANGE_START);
159
160     // Get storage slot for the range
161     Range storage range = _rangeByOfferId[_offerId];
162
163     // Revert if the offer id is already associated with a range
164     require(range.length == 0, OFFER_RANGE_ALREADY_RESERVED);
165
166     // Store the reserved range
167     range.offerId = _offerId;
168     range.start = _start;
169     range.length = _length;
170     _rangeOfferIds.push(_offerId);
171
172     emit RangeReserved(_offerId, range);
173 }
```

Recommendation:

While this is indirectly protected against via the maximum length limitation wherever `reserveRange` is invoked, it should also be validated locally as it is currently possible to create an invalid range entry which will be unusable due to an overflow failure.

Alleviation:

An overflow validation was introduced to the code via a `require` check ensuring that the `_start` of the range is less-than-or-equal to the maximum value that would disallow an overflow (i.e. `type(uint256).max - _length`). As a result, we consider this exhibit fully alleviated.

BVR-02M: Inexistent Validation of Range Length

Type	Severity	Location
Input Sanitization	Unknown	BosonVoucher.sol:L119, L155

Description:

The implementation of `issueVoucher` assumes that at least one member of the range will be reserved based on the first comparator of the newly introduced `require` check, however, the `reserveRange` function does not enforce a minimum `_length`. While the function is protected from an incorrect `_length` variable at the invocation level within `OfferBase`, it is also protected against an invalid `_start` even though it is validated locally.

Impact:

While currently not an active issue due to `reserveRange` being invoked by properly defined functions, this issue can arise in a future integration by another module of the Boson system invoking `reserveRange`.

Example:

contracts/protocol/clients/voucher/BosonVoucher.sol

SOL

```sol
/**
 * @notice Initializes the voucher.
 * This function is callable only once.
 */
function initializeVoucher(
    uint256 _sellerId,
    address _newOwner,
    VoucherInitValues calldata voucherInitValues
) public initializer {
    string memory sellerId = Strings.toString(_sellerId);
    string memory voucherName = string(abi.encodePacked(VOUCHER_NAME, " ", sellerId));
    string memory voucherSymbol = string(abi.encodePacked(VOUCHER_SYMBOL, "_", sellerI

    __ERC721_init_unchained(voucherName, voucherSymbol);

    // we don't call init on ownable, but rather just set the ownership to correct own
     transferOwnership( newOwner);
```

```
89
90      _setContractURI(voucherInitValues.contractURI);
91
92      _setRoyaltyPercentage(voucherInitValues.royaltyPercentage);
93
94      emit VoucherInitialized(_sellerId, _royaltyPercentage, _contractURI);
95  }
96
97  /**
98   * @notice Issues a voucher to a buyer.
99   *
100  * Minted voucher supply is sent to the buyer.
101  * Caller must have PROTOCOL role.
102  *
103  * Reverts if:
104  * - Exchange id falls within a reserved range
105  *
106  * @param _exchangeId - the id of the exchange (corresponds to the ERC-721 token id)
107  * @param _buyer - the buyer address
108  */
109 function issueVoucher(uint256 _exchangeId, address _buyer) external override onlyRole(
110     // Get the exchange
111     (, Exchange memory exchange) = getBosonExchange(_exchangeId);
112
113     // See if the offer id is associated with a range
114     Range storage range = _rangeByOfferId[exchange.offerId];
115
116     // Revert if exchange id falls within a reserved range
117     uint256 rangeStart = range.start;
118     require(
119         (_exchangeId < rangeStart) || (_exchangeId >= rangeStart + range.length),
120         EXCHANGE_ID_IN_RESERVED_RANGE
121     );
122
123     // Mint the voucher, sending it to the buyer address
124     _mint(_buyer, _exchangeId);
125 }
126
127 /**
128  * @notice Burns a voucher.
129  *
130  * Caller must have PROTOCOL role.
131  *
132  * @param _exchangeId - the id of the exchange (corresponds to the ERC-721 token id)
133  */
134 function burnVoucher(uint256 _exchangeId) external override onlyRole(PROTOCOL) {
135     _burn(_exchangeId);
```

```
136 }
137
138 /**
139  * @notice Reserves a range of vouchers to be associated with an offer
140  *
141  * Must happen prior to calling preMint
142  * Caller must have PROTOCOL role.
143  *
144  * Reverts if:
145  * - Start id is not greater than zero
146  * - Offer id is already associated with a range
147  *
148  * @param _offerId - the id of the offer
149  * @param _start - the first id of the token range
150  * @param _length - the length of the range
151  */
152 function reserveRange(
153     uint256 _offerId,
154     uint256 _start,
155     uint256 _length
156 ) external onlyRole(PROTOCOL) {
157     // Make sure range start is valid
158     require(_start > 0, INVALID_RANGE_START);
159
160     // Get storage slot for the range
161     Range storage range = _rangeByOfferId[_offerId];
162
163     // Revert if the offer id is already associated with a range
164     require(range.length == 0, OFFER_RANGE_ALREADY_RESERVED);
165
166     // Store the reserved range
167     range.offerId = _offerId;
168     range.start = _start;
169     range.length = _length;
170     _rangeOfferIds.push(_offerId);
171
172     emit RangeReserved(_offerId, range);
173 }
```

Recommendation:

We advise variable validation to be entirely relocated within the `BosonVoucher` to ensure self-sufficiency and a `require` check to be introduced ensuring a non-zero `_length` argument.

Alleviation:

A non-zero `_length` check has been adequately introduced in the `reserveRange` function, preventing unusable ranges from being defined.

BVR-03M: Weak Enforcement of `_rangeOfferIds` Ascending Order

Type	Severity	Location
Logical Fault	Unknown	BosonVoucher.sol:L152-L173

Description:

The `_rangeOfferIds` array entries should point to ranges that are in ascending order for the contract's binary search algorithm to iterate correctly. Instead of being upheld locally, this trait is upheld by `reserveRangeInternal` within `OfferBase` as well as `reserveRange` in `BosonVoucher` indirectly.

The first enforcement is inferred by how new reserved ranges are always in incremental fashion within `OfferOrder` while the latter enforcement occurs by the `reserveRange` function of `BosonVoucher` disallowing a range to be reserved again for the same offer ID.

Impact:

The binary search algorithm will cease functioning properly if the ascending order operational guarantee is not upheld which in turn would render premints inoperable. It is imperative that this type of guarantee is upheld locally due to the upgradeable nature of the Boson Protocol.

Example:

contracts/protocol/clients/voucher/BosonVoucher.sol

SOL

```
138 /**
139  * @notice Reserves a range of vouchers to be associated with an offer
140  *
141  * Must happen prior to calling preMint
142  * Caller must have PROTOCOL role.
143  *
144  * Reverts if:
145  * - Start id is not greater than zero
146  * - Offer id is already associated with a range
147  *
148  * @param _offerId - the id of the offer
149  * @param _start - the first id of the token range
150  * @param _length - the length of the range
```

```
151  */
152 function reserveRange(
153     uint256 _offerId,
154     uint256 _start,
155     uint256 _length
156 ) external onlyRole(PROTOCOL) {
157     // Make sure range start is valid
158     require(_start > 0, INVALID_RANGE_START);
159
160     // Get storage slot for the range
161     Range storage range = _rangeByOfferId[_offerId];
162
163     // Revert if the offer id is already associated with a range
164     require(range.length == 0, OFFER_RANGE_ALREADY_RESERVED);
165
166     // Store the reserved range
167     range.offerId = _offerId;
168     range.start = _start;
169     range.length = _length;
170     _rangeOfferIds.push(_offerId);
171
172     emit RangeReserved(_offerId, range);
173 }
```

Recommendation:

The security guarantee of an ascending order in the `_rangeOfferIds` should be validated locally rather than relying on proper sanitization by the caller as this type of validation delegation can lead to latent vulnerabilities. We advise the ascending order to be validated and enforced locally, ensuring the operational assumptions of the `BosonVoucher` are always upheld.

Alleviation:

The ascending order of the `_rangeOfferIds` array is now upheld locally via corresponding `require` checks that are conditional to the presence of existing ranges and validate the `_start` argument of the range.

BVR-04M: Potential Usability Enhancement of Preminted Vouchers

Type	Severity	Location
Standard Conformity	Minor	BosonVoucher.sol:L381-L399, L401-L420

Description:

The `transferFrom` and `safeTransferFrom` functions will execute an expensive binary search algorithm in the case of a preminted voucher which can incur significant gas and, in case multiple ranges have been defined for a particular voucher instance of a user, incur a significant gas overhead that could render "materializing" a preminted voucher unprofitable.

Impact:

Due to the expensive nature of on-chain computational resources, we classify this finding as minor based on the assumption that a user with multiple ranges will be dis-incentivized to issue preminted vouchers due to the significantly increased gas cost they would incur.

Example:

contracts/protocol/clients/voucher/BosonVoucher.sol

SOL

```
660 /**
661  * @dev Determines if a token is pre-minted and committable via transfer hook
662  *
663  * Committable means:
664  * - does not yet have an owner
665  * - in a reserved range
666  * - has been pre-minted
667  * - has not been already burned
668  *
669  * @param _tokenId - the token id to check
670  * @return committable - whether the token is committable
671  * @return offerId - the associated offer id if committable
672  */
673 function getPreMintStatus(uint256 _tokenId) internal view returns (bool committable, u
674     // Not committable if _committed already or if token has an owner
675     if (!_committed[_tokenId] && !_exists(_tokenId)) {
676         // If are reserved ranges, search them
```

```
677         uint256 length = _rangeOfferIds.length;
678         if (length > 0) {
679             // Binary search the ranges array
680             uint256 low = 0; // Lower bound of search (array index)
681             uint256 high = length; // Upper bound of search
682 
683             while (low < high) {
684                 // Calculate the current midpoint
685                 uint256 mid = (high + low) / 2;
686 
687                 // Get the range stored at the midpoint
688                 Range storage range = _rangeByOfferId[_rangeOfferIds[mid]];
689 
690                 // Get the beginning of the range once for reference
691                 uint256 start = range.start;
692 
693                 if (start > _tokenId) {
694                     // Split low and search again if target too high
695                     high = mid;
696                 } else if (start + range.minted - 1 >= _tokenId) {
697                     // Is token in target's minted range?
698 
699                     // It is committable if it has not been burned
700                     if (_tokenId > range.lastBurnedTokenId) {
701                         committable = true;
702                         offerId = range.offerId;
703                     }
704                     break; // Found!
705                 } else if (start + range.length - 1 >= _tokenId) {
706                     // No? Ok, is it in target's reserved range?
707                     committable = false;
708                     break; // Found!
709                 } else {
710                     // No? It may be in a higher range
711                     low = mid + 1;
712                 }
713             }
714         }
715     }
716 }
```

Recommendation:

Given that searching for the range an ID falls in is expensive but validating it is present in a particular range is simple, we advise a specialized `transferPremintedFrom` function to be introduced that accepts an

additional parameter indicating the `offerId` the `tokenId` is associated with. As the function is meant to be integrated by front-ends / off-chain services, it is possible to query the correct `offerId` off-chain via `getPreMintStatus` and pass it in to the function call significantly reducing the gas cost necessary to transfer a preminted voucher.

Alleviation:

A special `transferPremintedFrom` function was introduced to the codebase that accepts the `_offerId` a preminted `_tokenId` falls on and validates it prior to consuming it. As an additional step, the `getPreMintStatus` function was made `public` to ensure the off-chain workflow we advised can be implemented by the Boson Protocol team. As a result, the exhibit has been alleviated to the fullest extent possible by adhering to our recommendations.

BVR-05M: Incorrect Invocation of Voucher Transfer Hook

Type	Severity	Location
Logical Fault	Medium	BosonVoucher.sol:L652-L653

Description:

The `onVoucherTransferred` fallback in case the `_from` address is the `owner()` but the transfer of the NFT asset is not `committable` will invoke the `onVoucherTransferred` hook regardless of whether the `_to` address is zero or whether `_from` and `_to` are identical.

Impact:

Currently, it is possible to bypass the security guarantees of `onVoucherTransferred` and invoke it with incorrect arguments as well as during burn operations if the `owner` is the `_from` address.

Example:

contracts/protocol/clients/voucher/BosonVoucher.sol

SOL

```sol
631 function _beforeTokenTransfer(
632     address _from,
633     address _to,
634     uint256 _tokenId
635 ) internal override {
636     // Update the buyer associated with the voucher in the protocol
637     // Only when transferring, not minting or burning
638     if (_from == owner()) {
639         if (_premintStatus.committable) {
640             // Set the preminted token as committed
641             _committed[_tokenId] = true;
642 
643             // If this is a transfer of preminted token, treat it differently
644             address protocolDiamond = IClientExternalAddresses(BeaconClientLib._beacon
645             IBosonExchangeHandler(protocolDiamond).commitToPreMintedOffer(
646                 payable(_to),
647                 _premintStatus.offerId,
648                 _tokenId
649             );
```

```
650             delete _premintStatus;
651         } else {
652             // Already committed, treat as a normal transfer
653             onVoucherTransferred(_tokenId, payable(_to));
654         }
655     } else if (_from != address(0) && _to != address(0) && _from != _to) {
656         onVoucherTransferred(_tokenId, payable(_to));
657     }
658 }
```

Recommendation:

We advise the inner `if` case of the upper `if` clause validating the `owner` to be relocated to it (i.e. `if (_from == owner() && _premintStatus.committable)`), ensuring all conditions are correctly evaluated for the `onVoucherTransferred` hook execution. Additionally, given that `committable == true` infers that `silentMint` was invoked, the `_from` address is guaranteed to be the `owner()` in such a case. As such, the condition can be further optimized as only `_premintStatus.commitable == true`.

Alleviation:

The code was updated according to our recommendation, simplifying the `if` clause condition and correcting the conditional execution of the `onVoucherTransferred` hook.

DisputeResolverHandlerFacet Manual Review Findings

DRH-01M: Inexistent Escalation Incentive System

Type	Severity	Location
Logical Fault	Medium	DisputeResolverHandlerFacet.sol:L140, L462

Description:

The dispute resolvers in the latest iteration of the codebase are forced to apply a zero fee for providing their service. As a consequence, any escalation performed via the `DisputeHandlerFacet` will be free thus incentivizing users to immediately escalate their disputes at no cost.

Impact:

As all disputes would be immediately escalated at no cost, it can overwhelm dispute resolvers as well as imbalance the buyer-seller dynamic during disputes.

Example:

contracts/protocol/facets/DisputeResolverHandlerFacet.sol

SOL

```
139 // Protocol doesn't yet support DR fees
140 require(_disputeResolverFees[i].feeAmount == 0, FEE_AMOUNT_NOT_YET_SUPPORTED);
```

Recommendation:

We advise the escalation system to be revised in light of the removal of fees to ensure that the fair-play economical incentives of the system are upheld.

Alleviation:

The Boson Protocol team evaluated this exhibit and has opted to retain the current behaviour of the code in this iteration as they have no system to extract dispute resolver fees yet. They have opted for a permission-less approach with a vision to introduce non-zero fees in the future when pay-out systems will be

implemented. As a result of this, we consider this exhibit as acknowledged with the potential of a remediation being introduced in a future version.

ExchangeHandlerFacet Manual Review Findings

EHF-01M: Potential Fatal Failure of Voucher Redemption

Type	Severity	Location
Logical Fault	Medium	ExchangeHandlerFacet.sol:L448, L471, L810

Description:

The `redeemVoucher` function will invoke the `transferTwins` function which in turn will cause a dispute to be automatically raised if the sender is a contract via `raiseDisputeInternal`. Due to a revamp of the function in `DisputeBase`, it now will fail if the disputes region is paused (`disputesNotPaused` modifier). As a result, voucher redemption by smart contracts that could fail will be impossible to execute in case the caller is a contract.

Impact:

Due to the new changes introduced in the dispute raising mechanism, a smart contract which was previously coded with an assumption that its voucher redemption call will succeed when the dispute region is paused will now fail. Such an unhandled failure can lead to unredeemable vouchers as well as general misbehaviour at the integration level of the caller. It is important that upgrades to the protocol do not break expected behaviour of externally-integrating modules to ensure that integrations of Boson Protocol can properly equip their contracts without needing built-in upgradeable capabilities.

Example:

contracts/protocol/facets/ExchangeHandlerFacet.sol

SOL

```sol
807 if (transferFailed) {
808     // Raise a dispute if caller is a contract
809     if (sender.isContract()) {
810         raiseDisputeInternal(_exchange, _voucher, seller.id);
811     } else {
812         // Revoke voucher if caller is an EOA
813         revokeVoucherInternal(_exchange);
814         // N.B.: If voucher was revoked because transfer twin failed, then voucher was
815         shouldBurnVoucher = false;
```

```
816     }
817 }
```

Recommendation:

We advise the error to be handled gracefully via a `try-catch` structure that will cause the voucher revocation mechanism to kick in if a dispute cannot be raised due to the said region being paused.

Alleviation:

The Boson Protocol team opted to revert the code to its original behaviour as raising a dispute automatically for smart contract workflows is imperative to the protocol's proper operation. Given that the `DisputeBase::raiseDIsputeInternal` function's `disputesNotPaused` limitation was omitted, we consider this exhibit properly alleviated.

OfferBase Manual Review Findings

OBE-01M: Inexistent Integration of Unlimited Offers

Type	Severity	Location
Logical Fault	Minor	OfferBase.sol:L320

Description:

The Boson Protocol system supports a special flag for the `quantityAvailable` (`type(uint256).max`) meant to indicate that it should not be reduced when new vouchers are minted for it, however, this flag is not supported by the premint system.

Impact:

The current implementation of premints deviates from the original behaviour of standard voucher mints which is undesirable.

Example:

contracts/protocol/bases/OfferBase.sol

SOL

```sol
295 function reserveRangeInternal(uint256 _offerId, uint256 _length) internal {
296     // Get offer, make sure the caller is the operator
297     Offer storage offer = getValidOffer(_offerId);
298
299     // Prevent reservation of an empty range
300     require(_length > 0, INVALID_RANGE_LENGTH);
301
302     // Cannot reserve more than it's available
303     require(offer.quantityAvailable >= _length, INVALID_RANGE_LENGTH);
304
305     // Prevent reservation of too large range, since it affects exchangeId
306     require(_length < (1 << 128), INVALID_RANGE_LENGTH);
307
308     // Get starting token id
309     ProtocolLib.ProtocolCounters storage pc = protocolCounters();
310     uint256 _startId = pc.nextExchangeId;
```

```
312     // Call reserveRange on voucher
313     IBosonVoucher bosonVoucher = IBosonVoucher(protocolLookups().cloneAddress[offer.se
314     bosonVoucher.reserveRange(_offerId, _startId, _length);
315
316     // increase exchangeIds
317     pc.nextExchangeId = _startId + _length;
318
319     // decrease quantity available
320     offer.quantityAvailable -= _length;
321
322     // Notify external observers
323     emit RangeReserved(_offerId, offer.sellerId, _startId, _startId + _length - 1, msg
324 }
```

Recommendation:

We advise it to be done so, ensuring the premint system conforms to the specifications of the regular mint system for vouchers.

Alleviation:

The premint system now properly supports the infinite quantity flag, conditionally subtracting the `quantityAvailable` solely if it does not match the flag.

OBE-02M: Unsatisfied Function Pre-Conditions

Type	Severity	Location
Logical Fault	Minor	OfferBase.sol:L282, L283

Description:

The `reserveRangeInternal` function is meant to fail if the offers or exchanges region of the protocol is paused, however, no such validation is performed at the function level.

Impact:

It is currently possible to invoke `reserveRangeInternal` via multiple methods bypassing its security assumption. In detail, the exchange region assumption is invalidated by the following functions

Example:

contracts/protocol/bases/OfferBase.sol

SOL

```
278 /**
279  * @notice Reserves a range of vouchers to be associated with an offer
280  *
281  * Reverts if:
282  * - The offers region of protocol is paused
283  * - The exchanges region of protocol is paused
284  * - Offer does not exist
285  * - Offer already voided
286  * - Caller is not the seller
287  * - Range length is zero
288  * - Range length is greater than quantity available
289  * - Range length is greater than maximum allowed range length
290  * - Call to BosonVoucher.reserveRange() reverts
291  *
292  * @param _offerId - the id of the offer
293  * @param _length - the length of the range
294  */
295 function reserveRangeInternal(uint256 _offerId, uint256 _length) internal {
296     // Get offer, make sure the caller is the operator
297     Offer storage offer = getValidOffer(_offerId);
```

```
299     // Prevent reservation of an empty range
300     require(_length > 0, INVALID_RANGE_LENGTH);
301
302     // Cannot reserve more than it's available
303     require(offer.quantityAvailable >= _length, INVALID_RANGE_LENGTH);
304
305     // Prevent reservation of too large range, since it affects exchangeId
306     require(_length < (1 << 128), INVALID_RANGE_LENGTH);
307
308     // Get starting token id
309     ProtocolLib.ProtocolCounters storage pc = protocolCounters();
310     uint256 _startId = pc.nextExchangeId;
311
312     // Call reserveRange on voucher
313     IBosonVoucher bosonVoucher = IBosonVoucher(protocolLookups().cloneAddress[offer.se
314     bosonVoucher.reserveRange(_offerId, _startId, _length);
315
316     // increase exchangeIds
317     pc.nextExchangeId = _startId + _length;
318
319     // decrease quantity available
320     offer.quantityAvailable -= _length;
321
322     // Notify external observers
323     emit RangeReserved(_offerId, offer.sellerId, _startId, _startId + _length - 1, msg
324 }
```

Recommendation:

We advise the function to validate those two system modules being active as none of its internal or external call chains (`getValidOffer`, `protocolCounters`, `protocolLookups`, `reserveRange`) do so in the current implementation. While both regions are validated in the `reserveRange` function of `OfferHandlerFacet`, the same validation is not present in other invocations of the system such as within `OrchestrationHandlerFacet1`.

Alleviation:

The `offersNotPaused` and `exchangesNotPaused` modifiers have been introduced to the `reserveRangeInternal`, ensuring its pre-conditions are properly satisfied regardless of the workflow the function is invoked in.

ProtocolInitializationFacet Manual Review Findings

PIF-01M: Unprotected Facet w/ Delegate Call

Type	Severity	Location
Language Specific	Major	ProtocolInitializationFacet.sol:L54

Description:

The `ProtocolInitializationFacet` can perform `delegatecall` instructions and represents a logic implementation to which calls should be relayed to via the Diamond implementation. As the base implementation deployed for the Diamond to point to is unprotected, a malicious party can cause it to self-destruct rendering the `getVersion` function un-invoke-able.

Impact:

By self-destructing the version initializer, it would become impossible to query the current version via `getVersion` which future modules of Boson could rely on (such as automated version increment systems). Additionally, the module would be unusable in future version upgrades of the system requiring a re-deployment of the initializer.

Example:

contracts/protocol/facets/ProtocolInitializationFacet.sol

SOL

```
27  /**
28   * @notice Initializes the protocol after the deployment.
29   * This function is callable only once for each version
30   *
31   * @param _version - version of the protocol
32   * @param _addresses - array of facet addresses to call initialize methods
33   * @param _calldata -  array of facets initialize methods encoded as calldata
34   *                   _calldata order must match _addresses order
35   * @param _isUpgrade - flag to indicate whether this is first deployment or upgrade
36   * @param _initializationData - data for initialization of the protocol, using this fa
37   * @param _interfacesToRemove - array of interfaces to remove from the diamond
38   * @param _interfacesToAdd - array of interfaces to add to the diamond
39   */
```

```
function initialize(
    bytes32 _version,
    address[] calldata _addresses,
    bytes[] calldata _calldata,
    bool _isUpgrade,
    bytes calldata _initializationData,
    bytes4[] calldata _interfacesToRemove,
    bytes4[] calldata _interfacesToAdd
) external onlyUninitializedVersion(_version) {
    require(_version != bytes32(0), VERSION_MUST_BE_SET);
    require(_addresses.length == _calldata.length, ADDRESSES_AND_CALLDATA_LENGTH_MISMA

    // Delegate call to initialize methods of facets declared in _addresses
    for (uint256 i = 0; i < _addresses.length; i++) {
        (bool success, bytes memory error) = _addresses[i].delegatecall(_calldata[i]);

        // Handle result
        if (!success) {
            if (error.length > 0) {
                // bubble up the error
                revert(string(error));
            } else {
                // Reverts with default message
                revert(PROTOCOL_INITIALIZATION_FAILED);
            }
        }
    }

    ProtocolLib.ProtocolStatus storage status = protocolStatus();
    if (_isUpgrade) {
        if (_version == bytes32("2.2.0")) {
            initV2_2_0(_initializationData);
        }
    }

    removeInterfaces(_interfacesToRemove);
    addInterfaces(_interfacesToAdd);

    status.version = _version;
    emit ProtocolInitialized(string(abi.encodePacked(_version)));
}
```

Recommendation:

We advise a `constructor` to be introduced that sets an `immutable` variable with the value of `address(this)`. As a next step, we advise a `require` check to be introduced to `initialize` that ensures

only proxies invoke it by validating that `address(this) != immVar` where `immVar` represents the immutable variable we declared and assigned to in the previous step. This protection measure will prevent the `initialize` function from being invoked arbitrarily on the base implementation and thus prevent it from ever being self-destruct.

Alleviation:

The code was updated per our recommendation, introducing the `thisAddress` variable that is assigned during the contract's `constructor` and is consequently used to prevent direct invocations of the `initialize` call at the base implementation. As a result, destruction of the logic contract is not possible and thus the issue is considered adequately alleviated.

BosonVoucher Code Style Findings

BVR-01C: Duplication of Data Point

Type	Severity	Location
Gas Optimization	Informational	BosonVoucher.sol:L167

Description:

The `_offerId` data point is used as a key in the `_rangeByOfferId` mapping and is additionally stored within the `Range` struct.

Example:

contracts/protocol/clients/voucher/BosonVoucher.sol

SOL

```sol
function reserveRange(
    uint256 _offerId,
    uint256 _start,
    uint256 _length
) external onlyRole(PROTOCOL) {
    // Make sure range start is valid
    require(_start > 0, INVALID_RANGE_START);

    // Get storage slot for the range
    Range storage range = _rangeByOfferId[_offerId];

    // Revert if the offer id is already associated with a range
    require(range.length == 0, OFFER_RANGE_ALREADY_RESERVED);

    // Store the reserved range
    range.offerId = _offerId;
    range.start = _start;
    range.length = _length;
    _rangeOfferIds.push(_offerId);

    emit RangeReserved(_offerId, range);
}
```

Recommendation:

We advise it to either be omitted from the `Range` struct or to be filled in-memory whenever needed by callers as it is currently a redundant duplication of data as accessing the `Range` struct mandates knowledge of the `_offerId` it is associated with.

Alleviation:

The duplicated `offerId` data point has been safely omitted from the `Range` struct as per our recommendation.

BVR-02C: Generic Typographic Mistake

Type	Severity	Location
Code Style	Informational	BosonVoucher.sol:L69

Description:

The referenced line contains a typographical mistake or generic documentational error (i.e. copy-paste) that should be corrected.

Example:

contracts/protocol/clients/voucher/BosonVoucher.sol

SOL

```
69  // Tell is preminted voucher has already been _committed
```

Recommendation:

We advise this to be done so to enhance the legibility of the codebase.

Alleviation:

The typo has been properly corrected.

BVR-03C: Optimization of Binary Search Algorithm

Type	Severity	Location
Gas Optimization	Informational	BosonVoucher.sol:L696-L709

Description:

The custom binary search algorithm within `getPreMintStatus` evaluates whether each iteration of the search algorithm falls within the minted range of a particular reserved premint as well as within the reserved range of it.

Example:

contracts/protocol/clients/voucher/BosonVoucher.sol

SOL

```sol
673 function getPreMintStatus(uint256 _tokenId) internal view returns (bool committable, u
674     // Not committable if _committed already or if token has an owner
675     if (!_committed[_tokenId] && !_exists(_tokenId)) {
676         // If are reserved ranges, search them
677         uint256 length = _rangeOfferIds.length;
678         if (length > 0) {
679             // Binary search the ranges array
680             uint256 low = 0; // Lower bound of search (array index)
681             uint256 high = length; // Upper bound of search
682 
683             while (low < high) {
684                 // Calculate the current midpoint
685                 uint256 mid = (high + low) / 2;
686 
687                 // Get the range stored at the midpoint
688                 Range storage range = _rangeByOfferId[_rangeOfferIds[mid]];
689 
690                 // Get the beginning of the range once for reference
691                 uint256 start = range.start;
692 
693                 if (start > _tokenId) {
694                     // Split low and search again if target too high
695                     high = mid;
696                 } else if (start + range.minted - 1 >= _tokenId) {
697                     // Is token in target's minted range?
```

```
698
699                     // It is committable if it has not been burned
700                     if (_tokenId > range.lastBurnedTokenId) {
701                         committable = true;
702                         offerId = range.offerId;
703                     }
704                     break; // Found!
705                 } else if (start + range.length - 1 >= _tokenId) {
706                     // No? Ok, is it in target's reserved range?
707                     committable = false;
708                     break; // Found!
709                 } else {
710                     // No? It may be in a higher range
711                     low = mid + 1;
712                 }
713             }
714         }
715     }
716 }
```

Recommendation:

Given that the minted range of a reserved premint is a subset case of its reserved range, we advise only the latter to be evaluated and its inner body to be expanded to validate whether the ID that falls within the reserved range is preminted, thus reducing the complexity and operational cost of the binary search algorithm in its worst case scenario as well as significantly optimizing the best case scenario as only one member of the `range` struct would be read on each binary search iteration instead of both `minted` and `length`.

Alleviation:

The binary search algorithm has been significantly optimized by applying the recommended course of action and merging the in-range and in-mint range cases as they overlap.

DisputeHandlerFacet Code Style Findings

DHF-01C: Inefficient Application of Region Pause Status Enforcement

Type	Severity	Location
Gas Optimization	Informational	DisputeHandlerFacet.sol:L67, L166, L202, L235, L345, L381, L441, L443

Description:

The `disputesNotPaused` modifier is introduced throughout the functions of the `DisputeHandlerFacet`, however, this replication is unnecessary as it can be made optimal as well as more efficient.

Example:

contracts/protocol/facets/DisputeHandlerFacet.sol

SOL

```
152 /**
153  * @notice Expires the dispute and releases the funds.
154  *
155  * Emits a DisputeExpired event if successful.
156  *
157  * Reverts if:
158  * - The disputes region of protocol is paused
159  * - Exchange does not exist
160  * - Exchange is not in a Disputed state
161  * - Dispute is still valid
162  * - Dispute is in some state other than Resolving
163  *
164  * @param _exchangeId - the id of the associated exchange
165  */
166 function expireDispute(uint256 _exchangeId) public override disputesNotPaused nonReent
167     // Get the exchange, should be in disputed state
168     (Exchange storage exchange, ) = getValidExchange(_exchangeId, ExchangeState.Disput
169
170     // Fetch the dispute and dispute dates
171     (, Dispute storage dispute, DisputeDates storage disputeDates) = fetchDispute(_exc
172
```

```
173     // Make sure the dispute is in the resolving state
174     require(dispute.state == DisputeState.Resolving, INVALID_STATE);
175
176     // make sure the dispute not expired already
177     require(block.timestamp > disputeDates.timeout, DISPUTE_STILL_VALID);
178
179     // Finalize the dispute
180     finalizeDispute(_exchangeId, exchange, dispute, disputeDates, DisputeState.Retract
181
182     // Notify watchers of state change
183     emit DisputeExpired(_exchangeId, msgSender());
184 }
185
186 /**
187  * @notice Expires a batch of disputes and releases the funds.
188  *
189  * Emits a DisputeExpired event for every dispute if successful.
190  *
191  * Reverts if:
192  * - The disputes region of protocol is paused
193  * - Number of disputes exceeds maximum allowed number per batch
194  * - For any dispute:
195  *   - Exchange does not exist
196  *   - Exchange is not in a Disputed state
197  *   - Dispute is still valid
198  *   - Dispute is in some state other than Resolving
199  *
200  * @param _exchangeIds - the array of ids of the associated exchanges
201  */
202 function expireDisputeBatch(uint256[] calldata _exchangeIds) external override dispute
203     // limit maximum number of disputes to avoid running into block gas limit in a loo
204     require(_exchangeIds.length <= protocolLimits().maxDisputesPerBatch, TOO_MANY_DISP
205
206     for (uint256 i = 0; i < _exchangeIds.length; i++) {
207         // create offer and update structs values to represent true state
208         expireDispute(_exchangeIds[i]);
209     }
210 }
```

Recommendation:

The `disputesNotPaused` modifier can be relocated to the `finalizeDispute` function that is invoked by all current functions of the contract that apply the modifier (sans `extendDisputeTimeout` where it should remain). Additionally, in the current iteration the `disputesNotPaused` modifier is applied multiple times redundantly by the `expireDisputeBatch` function as it is validated once at the top level and consequently

at each `expireDispute` invocation. This would require a revamp of how the internal functions of the contract work to properly optimize.

Alleviation:

The code was updated to apply the `disputesNotPaused` modifier in the `finalizeDispute` call and omit it from all top-level functions that invoke it, optimizing the gas cost of each function in the process. While the internalization of how `expireDisputeBatch` and `expireDispute` would optimally operate was not performed, the Boson Protocol team stated that they will investigate this aspect more and potentially introduce it in a future iteration of the protocol. As a result, we consider this exhibit's proposed alleviation adequately applied.

ExchangeHandlerFacet Code Style Findings

EHF-01C: Redundant Duplication of Pre-Conditions

Type	Severity	Location
Code Style	Informational	ExchangeHandlerFacet.sol:L57, L58, L100, L101

Description:

Although two separate ways to mint an offer exist (pre-minted and normal), the documentation of pre-conditions for both is listed on each function.

Example:

contracts/protocol/facets/ExchangeHandlerFacet.sol

SOL

```
35  /**
36   * @notice Commits to an offer (first step of an exchange).
37   *
38   * Emits a BuyerCommitted event if successful.
39   * Issues a voucher to the buyer address.
40   *
41   * Reverts if:
42   * - The exchanges region of protocol is paused
43   * - The buyers region of protocol is paused
44   * - OfferId is invalid
45   * - Offer has been voided
46   * - Offer has expired
47   * - Offer is not yet available for commits
48   * - Offer's quantity available is zero
49   * - Buyer address is zero
50   * - Buyer account is inactive
51   * - Buyer is token-gated (conditional commit requirements not met or already used)
52   * - Offer price is in native token and caller does not send enough
53   * - Offer price is in some ERC20 token and caller also sends native currency
54   * - Contract at token address does not support ERC20 function transferFrom
55   * - Calling transferFrom on token fails for some reason (e.g. protocol is not approve
56   * - Received ERC20 token amount differs from the expected value
57   * - Seller has less funds available than sellerDeposit for non preminted offers
```

```
58   * - Seller has less funds available than sellerDeposit and price for preminted offers
59   *
60   * @param _buyer - the buyer's address (caller can commit on behalf of a buyer)
61   * @param _offerId - the id of the offer to commit to
62   */
63  function commitToOffer(address payable _buyer, uint256 _offerId)
64      external
65      payable
66      override
67      exchangesNotPaused
68      buyersNotPaused
69      nonReentrant
70  {
71      // Make sure buyer address is not zero address
72      require(_buyer != address(0), INVALID_ADDRESS);
73
74      // Get the offer
75      bool exists;
76      Offer storage offer;
77      (exists, offer) = fetchOffer(_offerId);
78
79      // Make sure offer exists, is available, and isn't void, expired, or sold out
80      require(exists, NO_SUCH_OFFER);
81
82      commitToOfferInternal(_buyer, offer, 0, false);
83  }
84
85  /**
86   * @notice Commits to a preminted offer (first step of an exchange).
87   *
88   * Emits a BuyerCommitted event if successful.
89   *
90   * Reverts if:
91   * - The exchanges region of protocol is paused
92   * - The buyers region of protocol is paused
93   * - Caller is not the voucher contract, owned by the seller
94   * - Exchange exists already
95   * - Offer has been voided
96   * - Offer has expired
97   * - Offer is not yet available for commits
98   * - Buyer account is inactive
99   * - Buyer is token-gated (conditional commit requirements not met or already used)
100  * - Seller has less funds available than sellerDeposit for non preminted offers
101  * - Seller has less funds available than sellerDeposit and price for preminted offers
102  *
103  * @param _buyer - the buyer's address (caller can commit on behalf of a buyer)
104  * @param _offerId - the id of the offer to commit to
105  * @param _exchangeId - the id of the exchange
```

```
106  */
107 function commitToPreMintedOffer(
108     address payable _buyer,
109     uint256 _offerId,
110     uint256 _exchangeId
111 ) external exchangesNotPaused buyersNotPaused nonReentrant {
112     // Fetch the offer info
113     (, Offer storage offer) = fetchOffer(_offerId);
114
115     // Make sure that the voucher was issued on the clone that is making a call
116     require(msg.sender == protocolLookups().cloneAddress[offer.sellerId], ACCESS_DENIE
117
118     // Exchange must not exist already
119     (bool exists, ) = fetchExchange(_exchangeId);
120     require(!exists, EXCHANGE_ALREADY_EXISTS);
121
122     commitToOfferInternal(_buyer, offer, _exchangeId, true);
123 }
```

Recommendation:

We advise each function to list its own pre-condition rather than both to avoid ambiguity and confusion in the way the function operates.

Alleviation:

The documentation was updated as per our advice, clearly defining the revert condition for each function.

OrchestrationHandlerFacet1 Code Style Findings

OH1-01C: Duplicate Enforcement of Pause Region Status

Type	Severity	Location
Gas Optimization	Informational	OrchestrationHandlerFacet1.sol:L934, L1023, L1121, L1223

Description:

The `createSellerAndPremintedOfferAndTwinWithBundle` and `createSellerAndPremintedOfferWithConditionAndTwinAndBundle` functions redundantly apply the `sellersNotPaused` modifier as it is applied by the functions `createSellerAndOfferAndTwinWithBundle` and `createSellerAndOfferWithConditionAndTwinAndBundle` that they respectively invoke.

Example:

contracts/protocol/facets/OrchestrationHandlerFacet1.sol

SOL

```
1211function createSellerAndPremintedOfferWithConditionAndTwinAndBundle(
1212    Seller memory _seller,
1213    Offer memory _offer,
1214    OfferDates calldata _offerDates,
1215    OfferDurations calldata _offerDurations,
1216    uint256 _disputeResolverId,
1217    uint256 _reservedRangeLength,
1218    Condition calldata _condition,
1219    Twin memory _twin,
1220    AuthToken calldata _authToken,
1221    VoucherInitValues calldata _voucherInitValues,
1222    uint256 _agentId
1223) external sellersNotPaused {
1224    createSellerAndOfferWithConditionAndTwinAndBundle(
1225        _seller,
1226        _offer,
1227        _offerDates,
1228        _offerDurations,
1229        _disputeResolverId,
1230        _condition,
1231        _twin,
```

```
1232        _authToken,
1233        _voucherInitValues,
1234        _agentId
1235    );
1236    reserveRangeInternal(_offer.id, _reservedRangeLength);
1237}
```

Recommendation:

We advise the modifier to be omitted from the preminted counterparts thus optimizing the gas cost of the codebase.

Alleviation:

The `sellersNotPaused` modifier was safely omitted from the two referenced functions, optimizing the gas cost of them and alleviating 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.

External Call Validation

Many contracts that interact with DeFi contain a set of complex external call executions that need to happen in a particular sequence and whose execution is usually taken for granted whereby it is not always the case. External calls should always be validated, either in the form of `require` checks imposed at the contract-level or via more intricate mechanisms such as invoking an external getter-variable and ensuring that it has been properly updated.

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 Solidity language boasts that discerns it from other conventional programming languages. For example, the EVM is a 256-bit machine meaning that operations on less-than-256-bit types are more costly for the EVM in terms of gas costs, meaning that loops utilizing a `uint8` variable because their limit will never exceed the 8-bit range actually cost more than redundantly using a `uint256` variable.

Code Style

An official Solidity style guide exists that is constantly under development and is adjusted on each new Solidity release, designating how the overall look and feel of a codebase should be. 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 contract-level variable that is present in the inheritance chain of the local execution level's context.

Gas Optimization

Gas optimization findings relate to ways the codebase can be optimized to reduce the gas cost involved with interacting with it to various degrees. These types of findings are completely optional and are pointed out for the benefit of the project's developers.

Standard Conformity

These types of findings relate to incompatibility between a particular standard's implementation and the project's implementation, oftentimes causing significant issues in the usability of the contracts.

Mathematical Operations

In Solidity, math generally behaves differently than other programming languages due to the constraints of the EVM. A prime example of this difference is the truncation of values during a division which in turn leads to loss of precision and can cause systems to behave incorrectly when dealing with percentages and proportion calculations.

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.

Centralization Concern

This category covers all findings that relate to a significant degree of centralization present in the project and as such the potential of a Single-Point-of-Failure (SPoF) for the project that we urge them to re-consider and potentially omit.

Reentrant Call

This category relates to findings that arise from re-entrant external calls (such as EIP-721 minting operations) and revolve around the inapplicacy of the Checks-Effects-Interactions (CEI) pattern, a pattern that dictates checks (`require` statements etc.) should occur before effects (local storage updates) and interactions (external calls) should be performed last.

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.