OMNISCIA

October 13, 2025

SMART CONTRACT AUDIT REPORT

Boson Protocol

Version 2.5.0

omniscia.io

info@omniscia.io

Online report: boson-protocol-version-2.5.0

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 550+ clients, detecting 2,500+ 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, AvaLabs, Gnosis, Euler Finance, Kyber Network, Stake Dao, Maverick Protocol, EtherFi, Karpatkey, OlympusDao, Morpho, Fetch_ai, Arcade_xyz, Gravita, LimitBreak, and many 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: boson-protocol-version-2.5.0

Version 2.5.0 Security Audit

Audit Report Revisions

Commit Hash	Date	Audit Report Hash
6a14c31bea	September 23rd 2025	ecbf71fe88
efd5d1a8f2	October 9th 2025	7b00478e71
73585830fd	October 10th 2025	7f9dab9c39
73585830fd	October 13th 2025	c5875fde65

Audit Overview

We were tasked with performing an audit of the Boson Protocol codebase and in particular the delta involved in their BPIP-8, BPIP-9, and BPIP-10 implementations.

BPIP-8 has introduced the concept of a mutualizer for dispute resolutions fees, permitting sellers to pay a premium to be covered by a dispute resolution fee mutualizer as a form of service.

BPIP-9 revolves around the introduction of buyer-initiated offers, ensuring that offer creations can occur by both sellers and buyers through slight refactors of the overall offer flows.

Finally, BPIP-10 relates to the introduction of off-chain offer agreements, permitting the offer to be created and executed in a single transaction through off-chain signatures.

Beyond the changes involved for the aforementioned Boson Protocol improvement proposals, we noticed that there have been several other slight refactors to the codebase.

Namely:

- Signature verification introduced **EIP-1271** support
- Optimizations in group validations
- Re-purposing of `FundsLib` from a library to a contract `FundsBase`
- Meta-transactions simplified through address appendation

Over the course of the audit, we identified vulnerabilities mostly around BPIP-8 with the changes involved in BPIP-9 and BPIP-10 being relatively straightforward.

In detail, we were able to identify a re-entrancy attack that breaches the security assumptions of a previously-deemed-safe function in `ExchangeCommitFacet` as well as a Denial-of-Service attack for fund releases through seller-defined mutualizers.

We advise the Boson 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 Boson 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 Boson Protocol and have identified that a particular exhibit has not been adequately dealt with. We advise the Boson Protocol team to revisit the following exhibit: `FBE-02M`

Post-Audit Conclusion (73585830fd)

The Boson Protocol team evaluated the final exhibit's addendum and proceeded with alleviating it fully.

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

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	0	0	0	0
Informational	12	9	0	3
Minor	5	5	0	0
Medium	0	0	0	0
Major	2	2	0	0

During the audit, we filtered and validated a total of **2 findings utilizing static analysis** tools as well as identified a total of **17 findings during the manual review** of the codebase. We strongly recommend that any minor severity or higher findings are dealt with promptly prior to the project's launch as they can introduce potential misbehaviours of the system as well as exploits.

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

Scope

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

Target

- Repository: **https://github.com/bosonprotocol/boson-protocol-contracts**
- Commit: 6a14c31bead936ed03b2d51ab82010ecd1047f40
- Language: Solidity
- Network: Ethereum, Polygon POS, Base, Optimism, Arbitrum, Boson, Fermion
- Revisions: **6a14c31bea**, **efd5d1a8f2**, **73585830fd**

Contracts Assessed

File	Total Finding(s)
contracts/protocol/facets/AgentHandlerFacet.sol (AHF)	0
contracts/protocol/bases/BuyerBase.sol (BBE)	0
contracts/domain/BosonTypes.sol (BTS)	0
contracts/protocol/bases/BundleBase.sol (BBS)	0
contracts/domain/BosonErrors.sol (BES)	0
contracts/protocol/clients/voucher/BosonVoucher.sol (BVR)	0
contracts/domain/BosonConstants.sol (BCS)	0
contracts/protocol/libs/BeaconClientLib.sol (BCL)	0
contracts/protocol/bases/BeaconClientBase.sol (BCB)	0
contracts/protocol/clients/proxy/BeaconClientProxy.sol (BCP)	0

contracts/protocol/facets/BuyerHandlerFacet.sol (BHF)	0
contracts/protocol/clients/priceDiscovery/BosonPriceDiscovery.sol (BPD)	0
contracts/protocol/libs/ClientLib.sol (CLB)	0
contracts/protocol/bases/ClientBase.sol (CBE)	0
contracts/protocol/clients/proxy/ClientProxy.sol (CPY)	0
contracts/protocol/facets/ConfigHandlerFacet.sol (CHF)	0
contracts/protocol/bases/ClientExternalAddressesBase.sol (CEA)	0
contracts/protocol/bases/DisputeBase.sol (DBE)	0
contracts/protocol/clients/DRFeeMutualizer.sol (DRF)	5
contracts/diamond/facets/DiamondCutFacet.sol (DCF)	0
contracts/protocol/facets/DisputeHandlerFacet.sol (DHF)	0
contracts/protocol/facets/DisputeResolverHandlerFacet.sol (DRH)	0
contracts/protocol/libs/EIP712Lib.sol (EIP)	1
contracts/diamond/facets/ERC165Facet.sol (ERC)	0

contracts/protocol/facets/ExchangeCommitFacet.sol (ECF)	5
contracts/protocol/facets/ExchangeHandlerFacet.sol (EHF)	0
contracts/protocol/bases/FundsBase.sol (FBE)	3
contracts/protocol/facets/FundsHandlerFacet.sol (FHF)	0
contracts/protocol/bases/GroupBase.sol (GBE)	0
contracts/protocol/facets/GroupHandlerFacet.sol (GHF)	0
contracts/protocol/facets/MetaTransactionsHandlerFacet.sol (MTH)	0
contracts/protocol/bases/OfferBase.sol (OBE)	3
contracts/protocol/facets/OfferHandlerFacet.sol (OHF)	1
contracts/protocol/facets/OrchestrationHandlerFacet1.sol (OH1)	0
contracts/protocol/facets/OrchestrationHandlerFacet2.sol (OH2)	0
contracts/protocol/libs/ProtocolLib.sol (PLB)	0
contracts/protocol/bases/ProtocolBase.sol (PBE)	1
contracts/protocol/facets/PauseHandlerFacet.sol (PHF)	0

contracts/protocol/bases/PriceDiscoveryBase.sol (PDB)	0
contracts/protocol/facets/PriceDiscoveryHandlerFacet.sol (PDH)	0
contracts/protocol/facets/ProtocolInitializationHandlerFacet.sol (PIH)	0
contracts/protocol/bases/SellerBase.sol (SBE)	0
contracts/example/SnapshotGate/SnapshotGate.sol (SGE)	0
contracts/protocol/facets/SellerHandlerFacet.sol (SHF)	0
contracts/protocol/facets/SequentialCommitHandlerFacet.sol (SCH)	0
contracts/protocol/bases/TwinBase.sol (TBE)	0
contracts/protocol/facets/TwinHandlerFacet.sol (THF)	0

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 `0.8.22` 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 located in external dependencies and can thus be safely ignored.

The `pragma` statements have been locked to `0.8.22` (`=0.8.22`), 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 **211 potential issues** within the codebase of which **209 were ruled out to be false positives** or negligible findings.

The remaining **2 issues** were validated and grouped and formalized into the **2 exhibits** that follow:

ID	Severity	Addressed	Title
DRF-01S	Informational	Yes	Inexistent Event Emission
OBE-01S	Informational	Yes	Literal Equality of `bool` Variable

Manual Review

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

As the audit at hand represents a code delta, intricate care was put into ensuring that the **previously established security guarantees remain upheld** and that **the new features that have been introduced to not result in retroactive vulnerabilities**.

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 multiple significant vulnerabilities** within the system which could have had **severe ramifications** to its overall operation; for more information, kindly consult the audit report's summary as well as the major exhibits contained within.

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 and explicit error cases for all newly introduced functions.

A total of **17 findings** were identified over the course of the manual review of which **8 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
DRF-01M	Informational	Yes	Improper Validation of Payment
EIP-01M	Minor	Yes	Limiting Signature Verification Process
ECF-01M	Minor	Yes	Incorrect Message Sender Emission
ECF-02M	Minor	Yes	Inexistent Validation of Mutualizer Interface Conformit
ECF-03M	Major	Yes	Fee Mutualizer Fund Extraction via Re-Entrancy

FBE-01M	Minor	Yes	Deprecated Approval Function
FBE-02M	Major	Yes	Insecure Integration of Dispute Resolution Mutualizer
OHF-01M	Minor	Yes	Inexistent Validation of Mutualizer Interface Conformity

Code Style

During the manual portion of the audit, we identified **9 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
DRF-01C	Informational	Yes	Code Repetition
DRF-02C	Informational	Yes	Inefficient `mapping` Lookups
DRF-03C	Informational	Yes	Redundant Parenthesis Statements
ECF-01C	Informational	Acknowledged	Redundant Conditional Validation
ECF-02C	Informational	Nullified	Redundant Validation of Seller Parameters
FBE-01C	Informational	Yes	Inefficient Handling of Dispute Resolution Refund
OBE-01C	Informational	Acknowledged	Redundant Conditional Validation
OBE-02C	Informational	Yes	Unspecified Revert Conditions
PBE-01C	Informational	Acknowledged	Redundant Conditional Validation

DRFeeMutualizer Static Analysis Findings

DRF-01S: Inexistent Event Emission

Type	Severity	Location
Language Specific	Informational	DRFeeMutualizer.sol:L487-L489

Description:

The linked function adjusts a sensitive contract variable yet does not emit an event for it.

Example:

contracts/protocol/clients/DRFeeMutualizer.sol

SOL

```sol
487 function setDepositRestriction(bool _restricted) external onlyOwner {
488     depositRestrictedToOwner = _restricted;
489 }
```

Recommendation:

We advise an `event` to be declared and correspondingly emitted to ensure off-chain processes can properly react to this system adjustment.

Alleviation (efd5d1a8f23c3bca7c25273ea4c912a367250119):

The `DepositRestrictionApplied` event was introduced to the codebase and is correspondingly emitted in the **`DRFeeMutualizer::setDepositRestriction`** function, addressing this exhibit in full.

OfferBase Static Analysis Findings

OBE-01S: Literal Equality of `bool` Variable

Type	Severity	Location
Gas Optimization	Informational	OfferBase.sol:L272

Description:

The linked `bool` comparison is performed between a variable and a `bool` literal.

Example:

contracts/protocol/bases/OfferBase.sol

SOL

```sol
272 if (!success || data.length != 32 || abi.decode(data, (bool)) == false) {
```

Recommendation:

We advise the `bool` variable to be utilized directly either in its negated (`!`) or original form.

Alleviation (efd5d1a8f23c3bca7c25273ea4c912a367250119):

The referenced boolean comparison has been optimized as advised.

DRFeeMutualizer Manual Review Findings

DRF-01M: Improper Validation of Payment

Type	Severity	Location
Logical Fault	Informational	DRFeeMutualizer.sol:L226

Description:

The `DRFeeMutualizer::returnDRFee` function will incorrectly permit an arbitrary amount of native funds to be sent to it if the `_returnedFeeAmount` is `0` due to not invoking the `FundsBase::validateIncomingPayment` function unconditionally.

Impact:

Although not a present issue, the lack of native amount validation in case a `_returnedFeeAmount` of `0` has been supplied can result in fund loss.

Example:

contracts/protocol/clients/DRFeeMutualizer.sol

SOL

```sol
216 function returnDRFee(
217     uint256 _exchangeId,
218     uint256 _returnedFeeAmount
219 ) external payable override onlyProtocol nonReentrant {
220     FeeInfo storage feeInfo = feeInfoByExchange[_exchangeId];
221     uint256 requestedFeeAmount = feeInfo.amount;
222     if (requestedFeeAmount == 0) revert InvalidExchangeId();
223
224     // Fee is being returned, add back to pool (if any)
225     if (_returnedFeeAmount > 0) {
```

Example (Cont.):

```sol
        validateIncomingPayment(feeInfo.token, _returnedFeeAmount);
        poolBalances[feeInfo.token] += _returnedFeeAmount;
    }

    delete feeInfoByExchange[_exchangeId];

    emit DRFeeReturned(_exchangeId, requestedFeeAmount, _returnedFeeAmount);
}
```

Recommendation:

We advise the `FundsBase::validateIncomingPayment` function to be invoked unconditionally, preventing accidental fund loss in future integrations.

Alleviation (efd5d1a8f23c3bca7c25273ea4c912a367250119):

The code was updated to no longer support native payments directly and to instead support their wrapped variant, alleviating this exhibit.

EIP712Lib Manual Review Findings

EIP-01M: Limiting Signature Verification Process

Type	Severity	Location
Logical Fault	Minor	EIP712Lib.sol:L50-L52, L69-L71, L73-L85

Description:

The `EIP712Lib::verify` function is meant to verify that a particular hashed message has been authorized either through **EIP-1271** or signature validation.

The current approach's error handling seems inconsistent, as the function implies a `revert` will occur if the `_user` is a contract that does not implement **EIP-1271** which is not the case.

Additionally, the implementation does not seem to account for smart accounts (EIP-7702) and will fatally `revert` in case **EIP-1271** is not adhered to strictly even though signature validation might ultimately succeed.

Impact:

The documentation presently implies a `revert` case that is unfulfilled, and will `revert` in certain edge cases that would otherwise succeed in signature validation of the previous implementation.

Example:

contracts/protocol/libs/EIP712Lib.sol

SOL

```
44  /**
45   * @notice Verifies that the signer really signed the message.
46   * It works for both ECDSA signatures and ERC1271 signatures.
47   *
48   * Reverts if:
49   * - Signer is the zero address
50   * - Signer is a contract that does not implement ERC1271
51   * - Signer is a contract that implements ERC1271 but returns an unexpected value
52   * - Signer is a contract that reverts when called with the signature
53   * - Signer is an EOA but the signature is not a valid ECDSA signature
```

Example (Cont.):

SOL

```sol
54   * - Recovered signer does not match the user address
55   *
56   * @param _user  - the message signer
57   * @param _hashedMessage - hashed message
58   * @param _signature - signature. If the signer is EOA, it must be ECDSA
signature in the format of (r,s,v) struct, otherwise, it must be a valid ERC1271
signature.
59   */
60  function verify(address _user, bytes32 _hashedMessage, bytes calldata _signature)
internal {
61      if (_user == address(0)) revert BosonErrors.InvalidAddress();
62
63      bytes32 typedMessageHash = toTypedMessageHash(_hashedMessage);
64
65      // Check if user is a contract implementing ERC1271
66      bytes memory returnData; // Make this available for later if needed
67      if (_user.code.length > 0) {
68          bool success;
69          (success, returnData) = _user.staticcall(
70              abi.encodeCall(IERC1271.isValidSignature, (typedMessageHash,
_signature))
71          );
72          if (success) {
73              if (returnData.length != SLOT_SIZE) {
74                  revert BosonErrors.UnexpectedDataReturned(returnData);
75              } else {
76                  // Make sure that the lowest 224 bits (28 bytes) are not set
77                  if (uint256(bytes32(returnData)) & type(uint224).max != 0) {
78                      revert BosonErrors.UnexpectedDataReturned(returnData);
79                  }
80
81                  if (abi.decode(returnData, (bytes4)) !=
IERC1271.isValidSignature.selector)
```

Example (Cont.):

SOL

```sol
                    revert BosonErrors.SignatureValidationFailed();

                return;
            }
        }
    }

    address signer;
    // If the user is not a contract or the call to ERC1271 failed, we assume it's an ECDSA signature
    if (_signature.length == 65) {
        ECDSASignature memory ecdsaSig = ECDSASignature({
            r: bytes32(_signature[0:32]),
            s: bytes32(_signature[32:64]),
            v: uint8(_signature[64])
        });

        // Ensure signature is unique
        // See https://github.com/OpenZeppelin/openzeppelin-contracts/blob/04695aecbd4d17dddfd55de766d10e3805d6f42f/contracts/cryptography/ECDSA.sol#63
        if (
            uint256(ecdsaSig.s) > 0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0 ||
            (ecdsaSig.v != 27 && ecdsaSig.v != 28)
        ) revert BosonErrors.InvalidSignature();

        signer = ecrecover(typedMessageHash, ecdsaSig.v, ecdsaSig.r, ecdsaSig.s);
        if (signer == address(0)) revert BosonErrors.InvalidSignature();
    }

    if (signer != _user) {
```

Example (Cont.):

SOL

```
110         if (returnData.length > 0) {
111             // In case 1271 verification failed with a revert reason, bubble it
up
112
113             /// @solidity memory-safe-assembly
114             assembly {
115                 revert(add(SLOT_SIZE, returnData), mload(returnData))
116             }
117         }
118
119         revert BosonErrors.SignatureValidationFailed();
120     }
121 }
```

Recommendation:

We advise the code to become **EIP-7702** conscious and to not `revert` fatally in its **EIP-1271** integration, updating its documented error cases in the process.

Alleviation (efd5d1a8f23c3bca7c25273ea4c912a367250119):

The code was updated to properly proceed with ECDSA validation regardless of the failure that was identified during the **EIP-1271** recovery flow, bubbling up the appropriate error should both validations fail.

As such, we consider this exhibit fully alleviated.

ExchangeCommitFacet Manual Review Findings

ECF-01M: Incorrect Message Sender Emission

Type	Severity	Location
Logical Fault	Minor	ExchangeCommitFacet.sol:L916

Description:

The `ExchangeCommitFacet::handleDRFeeCollection` function will incorrectly emit the `msg.sender` as the executor of the fee request whereas other event emissions will use the properly-evaluated `ProtocolBase::_msgSender`.

Impact:

The `DRFeeRequested` event will be emitted with a misleading executor data point in the current implementation.

Example:

contracts/protocol/facets/ExchangeCommitFacet.sol

SOL

```
910 // Emit event for DR fee request
911 emit IBosonFundsBaseEvents.DRFeeRequested(
912     _exchangeId,
913     exchangeToken,
914     _drFeeAmount,
915     _disputeTerms.mutualizerAddress,
916     msg.sender
917 );
```

Recommendation:

We advise the proper message sender to be resolved to ensure consistency across the codebase.

Alleviation (efd5d1a8f23c3bca7c25273ea4c912a367250119):

The event emission was updated to properly utilize the `ProtocolBase::_msgSender`, addressing this exhibit.

ECF-02M: Inexistent Validation of Mutualizer Interface Conformity

Type	Severity	Location
Input Sanitization	Minor	ExchangeCommitFacet.sol:L962

Description:

In contrast to the `OfferBase::storeOffer` function, the `ExchangeCommitFacet::addSellerParametersToBuyerOffer` function does not validate that the `_sellerParams.mutualizerAddress` adheres to the expected mutualizer interface.

Impact:

A non-functional mutualizer can be presently configured in the `ExchangeCommitFacet::addSellerParametersToBuyerOffer` function.

Example:

contracts/protocol/facets/ExchangeCommitFacet.sol

SOL

```
934 function addSellerParametersToBuyerOffer(
935     address _committer,
936     uint256 _offerId,
937     SellerOfferParams calldata _sellerParams
938 ) internal returns (BosonTypes.Offer storage offer) {
939     offer = getValidOffer(_offerId);
940     if (offer.priceType != PriceType.Static) revert InvalidPriceType();
941     if (offer.creator != OfferCreator.Buyer) revert InvalidOfferCreator();
942
943     (bool sellerExists, uint256 sellerId) = getSellerIdByAssistant(_committer);
```

Example (Cont.):

```sol
944     if (!sellerExists) revert NotAssistant();
945     offer.sellerId = sellerId;
946
947     {
948         ProtocolLib.ProtocolLookups storage lookups = protocolLookups();
949         if (_sellerParams.collectionIndex > 0) {
950             if (lookups.additionalCollections[sellerId].length <
_sellerParams.collectionIndex) {
951                 revert NoSuchCollection();
952             }
953             offer.collectionIndex = _sellerParams.collectionIndex;
954         }
955
956         validateRoyaltyInfo(lookups, protocolLimits(), sellerId,
_sellerParams.royaltyInfo);
957
958         offer.royaltyInfo[0] = _sellerParams.royaltyInfo;
959
960         if (_sellerParams.mutualizerAddress != address(0)) {
961             DisputeResolutionTerms storage disputeTerms =
fetchDisputeResolutionTerms(_offerId);
962             disputeTerms.mutualizerAddress = _sellerParams.mutualizerAddress;
963         }
964
965         emit BuyerInitiatedOfferSetSellerParams(_offerId, sellerId,
_sellerParams, _committer);
966     }
967 }
```

Recommendation:

We advise such validation to be imposed, ensuring the updated mutualizer satisfies the expected interface.

Alleviation (efd5d1a8f23c3bca7c25273ea4c912a367250119):

The mutualizer interface validation has been introduced to the `ExchangeCommitFacet::addSellerParametersToBuyerOffer` function as advised.

ECF-03M: Fee Mutualizer Fund Extraction via Re-Entrancy

Type	Severity	Location
Logical Fault	Major	**ExchangeCommitFacet.sol:L472**, **L554**, **L637**, **L664**, **L890**, **I**

Description:

The BPIP-8 standard introduced to the Boson Protocol system involves a novel dispute resolution fee mutualizer implementation meant to facilitate the provision of dispute resolution fees at a premium instead of forcing buyers or sellers to contribute them.

A complex re-entrancy attack has been introduced as a result of this feature due to how the dispute resolution fee is requested and captured.

Specifically, the `ExchangeCommitFacet::handleDRFeeCollection` function will measure the balance before and after the `IDRFeeMutualizer::requestDRFee` function call. A re-entrancy would permit the same balance delta (i.e. single transfer) to be measured as if multiple transfers occurred, thereby crediting an arbitrary number `n` of dispute resolution fees through a single transfer.

While most of the codebase is protected against re-entrancy attacks, a previously-established security assumption is now breached in the `ExchangeCommitFacet::onPremintedVoucherTransferred` function.

As the function is not protected against re-entrancy guards, it is possible to execute the described re-entrancy attack through the function via single user controlled buyers and sellers.

As multiple dispute resolution fees would be credited through a single transfer, fund extraction is possible by simply refunding all credited dispute resolution fees through the `FundsBase::releaseFunds` path.

Impact:

A re-entrancy edge case has been introduced in the Boson Protocol system through the BPIP-8 implementation that can result in fund loss for the protocol.

Example:

contracts/protocol/facets/ExchangeCommitFacet.sol

SOL

```sol
// Use mutualizer: request fee
uint256 balanceBefore = getBalance(exchangeToken);

// Request DR fee from mutualizer
bool success = IDRFeeMutualizer(mutualizer).requestDRFee(
    _offer.sellerId,
    _drFeeAmount,
    exchangeToken,
    _exchangeId,
    _disputeTerms.disputeResolverId
```

Example (Cont.):

```sol
899 );
900
901 uint256 balanceAfter = getBalance(exchangeToken);
```

Recommendation:

There are multiple approaches in addressing this particular issue, with one being the validation of mutualizer addresses against a list of authorized implementations.

An alternative approach would be to introduce an "atomic" re-entrancy guard to the `ExchangeCommitFacet::handleDRFeeCollection` function block that prevents this particular function from being re-entered.

Alleviation (efd5d1a8f23c3bca7c25273ea4c912a367250119):

A localized re-entrancy guard was introduced to the `ExchangeCommitFacet::onPremintedVoucherTransferred` function preventing re-entrancies from triggering automatic commitments to offers more than once in a single transfer, thereby eliminating the re-entrancy risk outlined and thus alleviating this exhibit.

FundsBase Manual Review Findings

FBE-01M: Deprecated Approval Function

Type	Severity	Location
Standard Conformity	Minor	FundsBase.sol:L275

Description:

The `FundsBase::releaseFunds` function will issue a `SafeERC20::safeApprove` function invocation of OpenZeppelin's `v4.9.6` release which is officially deprecated.

Impact:

Although unlikely, any approval by the `FundsBase::releaseFunds` implementation that is not consumed in full can cause fund release revert errors.

Example:

contracts/protocol/bases/FundsBase.sol

SOL

```
275 IERC20(exchangeToken).safeApprove(exchange.mutualizerAddress, returnAmount);
```

Recommendation:

We advise the code to utilize `SafeERC20::forceApprove` instead, preventing any lingering approvals from causing fatal `revert` errors.

Alleviation (efd5d1a8f23c3bca7c25273ea4c912a367250119):

The code was updated to set the allowance of the mutualizer via the `SafeERC20::forceApprove` function, addressing this exhibit.

FBE-02M: Insecure Integration of Dispute Resolution Mutualizer

Type	Severity	Location
Logical Fault	Major	FundsBase.sol:L267-L279

Description:

The `FundsBase::releaseFunds` function will interact with the `IDRFeeMutualizer` regardless of whether an actual refund is meant to be performed as it will notify the `DRFeeMutualizer` even with an argument of `0` as the `_returnedFeeAmount`.

A malicious `mutualizer` can induce a forcible `revert`, freezing any attempt to release funds for a particular exchange.

Impact:

A malicious mutualizer can induce errors in fund releases for Boson Protocol exchanges.

Example:

contracts/protocol/bases/FundsBase.sol

SOL

```sol
252 // Return unused DR fee to mutualizer or seller's pool
253 if (drTerms.feeAmount != 0) {
254     uint256 returnAmount = drTerms.feeAmount - payoff.disputeResolver;
255
256     // Use exchange-level mutualizer address (locked at commitment time)
257     if (exchange.mutualizerAddress == address(0)) {
258         if (returnAmount > 0) {
259             increaseAvailableFundsAndEmitEvent(
260                 _exchangeId,
261                 offer.sellerId,
```

Example (Cont.):

SOL

```
                exchangeToken,
                returnAmount,
                sender
            );
        }
    } else {
        if (exchangeToken == address(0)) {
            IDRFeeMutualizer(exchange.mutualizerAddress).returnDRFee{ value: returnAmount }(
                _exchangeId,
                returnAmount
            );
        } else {
            if (returnAmount > 0) {
                IERC20(exchangeToken).safeApprove(exchange.mutualizerAddress, returnAmount);
            }
            IDRFeeMutualizer(exchange.mutualizerAddress).returnDRFee(_exchangeId, returnAmount);
        }
    }
```

Recommendation:

We advise the code to ensure that the `DRFeeMutualizer` interaction is performed as securely as possible, ensuring that `revert` errors are ignored (or logged for a retry), and that a sensible gas stipend is provided to the `mutualizerAddress` to ensure that a Denial-of-Service attack through gas bombing cannot be attempted.

Alleviation (efd5d1a8f2):

The code was updated by handling a `revert` in the `IDRFeeMutualizer::finalizeExchange` call as well as allocating a fixed gas stipend to it.

We consider a non-dynamic gas stipend to be significantly restrictive as mutualizer implementations may change and the EVMs the project is deployed on may change as well, and we advise a dynamically configurable value to be utilized instead.

Alleviation (73585830fd):

The Boson Protocol team introduced functions to administratively configure the gas allocation of fee mutualizer calls, fully alleviating this exhibit.

OfferHandlerFacet Manual Review Findings

OHF-01M: Inexistent Validation of Mutualizer Interface Conformity

Type	Severity	Location
Input Sanitization	Minor	OfferHandlerFacet.sol:L514

Description:

In contrast to the `OfferBase::storeOffer` function, the `OfferHandlerFacet::updateOfferMutualizerInternal` function does not validate that the `_newMutualizer` adheres to the expected mutualizer interface.

Impact:

A non-functional mutualizer can be presently configured in the `OfferHandlerFacet::updateOfferMutualizerInternal` function.

Example:

contracts/protocol/facets/OfferHandlerFacet.sol

SOL

```
508 function updateOfferMutualizerInternal(uint256 _offerId, address _newMutualizer)
internal {
509     // Make sure the caller is the assistant, offer exists and is not voided
510     Offer storage offer = getValidOfferWithSellerCheck(_offerId);
511
512     DisputeResolutionTerms storage disputeResolutionTerms =
fetchDisputeResolutionTerms(_offerId);
513     if (disputeResolutionTerms.mutualizerAddress == _newMutualizer) revert
SameMutualizerAddress();
514     disputeResolutionTerms.mutualizerAddress = payable(_newMutualizer);
515
516     emit OfferMutualizerUpdated(_offerId, offer.sellerId, _newMutualizer,
_msgSender());
517 }
```

Recommendation:

We advise such validation to be imposed, ensuring the updated mutualizer satisfies the expected interface.

Alleviation (efd5d1a8f23c3bca7c25273ea4c912a367250119):

The code was updated to properly validate the mutualizer interface during an update, alleviating this exhibit.

DRFeeMutualizer Code Style Findings

DRF-01C: Code Repetition

Type	Severity	Location
Gas Optimization	Informational	**DRFeeMutualizer.sol**: • I-1: **L255-L260** • I-2: **L465-L470**

Description:

The referenced code block is implemented by the imported `FundsBase::validateIncomingPayment` function.

Example:

contracts/protocol/clients/DRFeeMutualizer.sol

SOL

```sol
250 function deposit(address _tokenAddress, uint256 _amount) external payable
nonReentrant {
251     address msgSender = _msgSender();
252     if (depositRestrictedToOwner && msgSender != owner()) revert
DepositsRestrictedToOwner();
253     if (_amount == 0) revert InvalidAmount();
254
255     if (_tokenAddress == address(0)) {
256         if (msg.value != _amount) revert BosonErrors.InsufficientValueReceived();
257     } else {
258         if (msg.value != 0) revert BosonErrors.NativeNotAllowed();
259         transferFundsIn(_tokenAddress, msgSender, _amount);
```

Example (Cont.):

```sol
    }

    poolBalances[_tokenAddress] += _amount;
    emit FundsDeposited(msgSender, _tokenAddress, _amount);
}
```

Recommendation:

We advise the function to be invoked, optimizing the code's syntax.

Alleviation (efd5d1a8f23c3bca7c25273ea4c912a367250119):

The code was updated to utilize the `FundsBase::validateIncomingPayment` function correctly, addressing this exhibit.

DRF-02C: Inefficient `mapping` Lookups

Type	Severity	Location
Gas Optimization	Informational	**DRFeeMutualizer.sol**: • I-1: **L135**, **L138** • I-2: **L182**, **L185** • I-3: **L342-L344**, **L379** • I-4: **L518**, **L521**

Description:

The linked statements perform key-based lookup operations on `mapping` declarations from storage multiple times for the same key redundantly.

Example:

contracts/protocol/clients/DRFeeMutualizer.sol

SOL

```
182 uint256 agreementId = sellerToTokenToDisputeResolverToAgreement[_sellerId]
[_tokenAddress][_disputeResolverId];
183 if (agreementId == 0 && _disputeResolverId != 0) {
184     // If no specific agreement exists, check for "any dispute resolver"
agreement
185     agreementId = sellerToTokenToDisputeResolverToAgreement[_sellerId]
[_tokenAddress][0];
```

Recommendation:

As the lookups internally perform an expensive `keccak256` operation, we advise the lookups to be cached wherever possible to a single local declaration that either holds the value of the `mapping` in case of primitive types or holds a `storage` pointer to the `struct` contained.

As the compiler's optimizations may take care of these caching operations automatically at-times, we advise the optimization to be selectively applied, tested, and then fully adopted to ensure that the proposed caching model indeed leads to a reduction in gas costs.

Alleviation (efd5d1a8f23c3bca7c25273ea4c912a367250119):

All `mapping` lookup pairs highlighted have been optimized per our recommendation.

DRF-03C: Redundant Parenthesis Statements

Type	Severity	Location
Code Style	Informational	**DRFeeMutualizer.sol**: • I-1: **L412** • I-2: **L413**

Description:

The referenced statements are redundantly wrapped in parenthesis' (`()`).

Example:

contracts/protocol/clients/DRFeeMutualizer.sol

SOL

```
412 (agreement.startTime > 0 && agreement.refundOnCancel) &&
```

Recommendation:

We advise them to be safely omitted, increasing the legibility of the codebase.

Alleviation (efd5d1a8f23c3bca7c25273ea4c912a367250119):

The redundant parenthesis in the referenced statements have been safely omitted.

ExchangeCommitFacet Code Style Findings

ECF-01C: Redundant Conditional Validation

Type	Severity	Location
Gas Optimization	Informational	ExchangeCommitFacet.sol:L374

Description:

The referenced conditional will redundantly evaluate whether the `OfferCreator` enum of the `Offer` structure is a `Buyer` even though the `enum` supports two states and one of the two has already been excluded.

Example:

contracts/protocol/facets/ExchangeCommitFacet.sol

SOL

```sol
368 if (_fullOffer.offer.creator == OfferCreator.Seller) {
369     // Validate caller is seller assistant
370     (, uint256 sellerId) = getSellerIdByAssistant(_offerCreator);
371     if (sellerId != _fullOffer.offer.sellerId) {
372         revert NotAssistant();
373     }
374 } else if (_fullOffer.offer.creator == OfferCreator.Buyer) {
375     uint256 buyerId = getValidBuyer(payable(_offerCreator));
376     if (buyerId != _fullOffer.offer.buyerId) {
377         revert NotBuyerWallet();
```

Example (Cont.):

```sol
    }
}
```

Recommendation:

We advise the referenced conditional to be set to an `else` clause, optimizing its gas cost.

Alleviation (efd5d1a8f23c3bca7c25273ea4c912a367250119):

The Boson Protocol team evaluated this exhibit and opted to retain the code as is as they consider it to be more legible and future-proof.

ECF-02C: Redundant Validation of Seller Parameters

Type	Severity	Location
Gas Optimization	Informational	ExchangeCommitFacet.sol:L260-L266

Description:

The `ExchangeCommitFacet::createOfferAndCommit` function will validate the `BosonTypes.SellerOfferParams` if the offer creator is the `Seller` to ensure they remain unspecified, however, they will not be utilized in such a case.

Example:

contracts/protocol/facets/ExchangeCommitFacet.sol

SOL

```
260 if (
261     _fullOffer.offer.creator == BosonTypes.OfferCreator.Seller &&
262     (_sellerParams.collectionIndex != 0 ||
263         _sellerParams.royaltyInfo.recipients.length != 0 ||
264         _sellerParams.royaltyInfo.bps.length != 0 ||
265         _sellerParams.mutualizerAddress != address(0))
266 ) revert SellerParametersNotAllowed();
267
268 // verify signature and potential cancellation
269 (bytes32 offerHash, uint256 offerId) = verifyOffer(_fullOffer, _offerCreator,
_signature);
```

Example (Cont.):

```sol

// create an offer
if (offerId == 0) {
    offerId = createOfferInternal(
        _fullOffer.offer,
        _fullOffer.offerDates,
        _fullOffer.offerDurations,
        _fullOffer.drParameters,
        _fullOffer.agentId,
        _fullOffer.feeLimit,
        false
    );
    protocolLookups().offerIdByHash[offerHash] = offerId;

    if (_fullOffer.condition.method != BosonTypes.EvaluationMethod.None) {
        // Construct new group
        // - group id is 0, and it is ignored
        Group memory group;
        group.sellerId = _fullOffer.offer.sellerId;
        group.offerIds = new uint256[](1);
        group.offerIds[0] = offerId;

        // Create group and update structs values to represent true state
        createGroupInternal(group, _fullOffer.condition, false);
    }
}

// Deposit other committer's funds if needed
```

Example (Cont.):

```sol
if (!_fullOffer.useDepositedFunds) {
    uint256 offerCreatorId;
    uint256 offerCreatorAmount;
    if (_fullOffer.offer.creator == BosonTypes.OfferCreator.Buyer) {
        // Buyer-created offer: committer is the seller
        offerCreatorId = _fullOffer.offer.buyerId;
        offerCreatorAmount = _fullOffer.offer.price;
    } else {
        // Seller-created offer: committer is the buyer
        offerCreatorId = _fullOffer.offer.sellerId;
        offerCreatorAmount = _fullOffer.offer.sellerDeposit;
    }

    if (offerCreatorAmount > 0) {
        transferFundsIn(_fullOffer.offer.exchangeToken, _offerCreator,
offerCreatorAmount);
        increaseAvailableFunds(offerCreatorId, _fullOffer.offer.exchangeToken,
offerCreatorAmount);
        emit IBosonFundsEvents.FundsDeposited(
            offerCreatorId,
            _offerCreator,
            _fullOffer.offer.exchangeToken,
            offerCreatorAmount
        );
    }
}

if (_fullOffer.condition.method != BosonTypes.EvaluationMethod.None) {
    if (_fullOffer.offer.creator == BosonTypes.OfferCreator.Buyer) {
        addSellerParametersToBuyerOffer(_committer, offerId, _sellerParams);
```

Example (Cont.):

```sol
    }

    commitToConditionalOffer(_committer, offerId, _conditionalTokenId);
} else {
    if (_fullOffer.offer.creator == BosonTypes.OfferCreator.Buyer) {
        commitToBuyerOffer(offerId, _sellerParams);
    } else {
        commitToOffer(_committer, offerId);
    }
}
```

Recommendation:

We advise this validation to be omitted as it restricts the function's input arguments redundantly.

Alleviation (efd5d1a8f23c3bca7c25273ea4c912a367250119):

The Boson Protocol team evaluated this exhibit and clarified that they wish to validate the presence of default values so as to avoid a buyer assuming their configured values have taken effect.

As such, we consider this exhibit to be nullified given that the original behaviour is explicitly desirable.

FundsBase Code Style Findings

FBE-01C: Inefficient Handling of Dispute Resolution Refund

Type	Severity	Location
Gas Optimization	Informational	FundsBase.sol:L253, L254, L258, L274

Description:

The BPIP-8 has introduced a specialized mutualizer address that is meant to handle dispute resolution fee provisions as well as refunds, and this contract has been integrated by the `FundsBase::releaseFunds` finalization action.

The current integration appears to be inefficient due to two points; firstly, the `DisputeResolutionTerms` data points are read multiple times inefficiently from `storage` whereas they should be loaded to `memory`.

Secondly, the current mechanism will either issue a total refund or no refund at all.

Example:

contracts/protocol/bases/FundsBase.sol

SOL

```
252 // Return unused DR fee to mutualizer or seller's pool
253 if (drTerms.feeAmount != 0) {
254     uint256 returnAmount = drTerms.feeAmount - payoff.disputeResolver;
255
256     // Use exchange-level mutualizer address (locked at commitment time)
257     if (exchange.mutualizerAddress == address(0)) {
258         if (returnAmount > 0) {
259             increaseAvailableFundsAndEmitEvent(
260                 _exchangeId,
261                 offer.sellerId,
```

Example (Cont.):

SOL

```
                exchangeToken,
                returnAmount,
                sender
            );
        }
    } else {
        if (exchangeToken == address(0)) {
            IDRFeeMutualizer(exchange.mutualizerAddress).returnDRFee{ value:
returnAmount }(
                _exchangeId,
                returnAmount
            );
        } else {
            if (returnAmount > 0) {
                IERC20(exchangeToken).safeApprove(exchange.mutualizerAddress,
returnAmount);
            }
            IDRFeeMutualizer(exchange.mutualizerAddress).returnDRFee(_exchangeId,
returnAmount);
        }
    }
}
```

Recommendation:

We advise the code to be optimized by loading data into `memory` and by optimizing the `if` conditional that triggers the refund to `drTerms.feeAmount != 0 && payoff.disputeResolver == 0`, permitting the code within to omit several conditionals.

Alleviation (efd5d1a8f23c3bca7c25273ea4c912a367250119):

The exhibit was partly addressed by introducing the `memory` related optimization, however, the Boson Protocol team clarified that zero-value exchange finalizations on the `IDRFeeMutualizer` should occur to ensure it is informed that no fee was ultimately captured.

As such, we consider this exhibit fully addressed.

OfferBase Code Style Findings

OBE-01C: Redundant Conditional Validation

Type	Severity	Location
Gas Optimization	Informational	OfferBase.sol:L115

Description:

The referenced conditional will redundantly evaluate whether the `OfferCreator` enum of the `Offer` structure is a `Buyer` even though the `enum` supports two states and one of the two has already been excluded.

Example:

contracts/protocol/bases/OfferBase.sol

SOL

```sol
104 if (_offer.creator == OfferCreator.Seller) {
105     if (_offer.buyerId != 0) revert InvalidBuyerOfferFields();
106
107     // Validate caller is seller assistant
108     if (_authenticate) {
109         (bool isAssistant, uint256 sellerId) = getSellerIdByAssistant(sender);
110         if (!isAssistant) {
111             revert NotAssistant();
112         }
113         _offer.sellerId = sellerId;
```

Example (Cont.):

```sol
    }
} else if (_offer.creator == OfferCreator.Buyer) {
```

Recommendation:

We advise the referenced conditional to be set to an `else` clause, optimizing its gas cost.

Alleviation (efd5d1a8f23c3bca7c25273ea4c912a367250119):

The Boson Protocol team evaluated this exhibit and opted to retain the code as is as they consider it to be more legible and future-proof.

OBE-02C: Unspecified Revert Conditions

Type	Severity	Location
Code Style	Informational	OfferBase.sol:L122, L123

Description:

The referenced two conditions represent `revert` cases that are undocumented.

Example:

contracts/protocol/bases/OfferBase.sol

SOL

```sol
/**
 * @notice Creates offer. Can be reused among different facets.
 *
 * Emits an OfferCreated event if successful.
 *
 * Reverts if:
 * - Caller is not an assistant in case of seller-initiated offer
 * - sellerId is not 0 when buyer-initiated offer is created
 * - collectionIndex is not 0 when buyer-initiated offer is created
 * - royaltyInfo is not empty when buyer-initiated offer is created
```

Example (Cont.):

```sol
60   * - priceType is not Static when buyer-initiated offer is created
61   * - Invalid offer creator value specified (OfferCreator.Seller or
OfferCreator.Buyer)
62   * - Valid from date is greater than valid until date
63   * - Valid until date is not in the future
64   * - Both voucher expiration date and voucher expiration period are defined
65   * - Neither of voucher expiration date and voucher expiration period are defined
66   * - Voucher redeemable period is fixed, but it ends before it starts
67   * - Voucher redeemable period is fixed, but it ends before offer expires
68   * - Dispute period is less than minimum dispute period
69   * - Resolution period is not between the minimum and the maximum resolution
period
70   * - Voided is set to true
71   * - Available quantity is set to zero
72   * - Dispute resolver wallet is not registered, except for absolute zero offers
with unspecified dispute resolver
73   * - Dispute resolver is not active, except for absolute zero offers with
unspecified dispute resolver
74   * - Seller is not on dispute resolver's seller allow list
75   * - Dispute resolver does not accept fees in the exchange token
76   * - Buyer cancel penalty is greater than price
77   * - When agent id is non zero:
78   *   - If Agent does not exist
79   * - If the sum of agent fee amount and protocol fee amount is greater than the
offer fee limit determined by the protocol
80   * - If the sum of agent fee amount and protocol fee amount is greater than fee
limit set by seller
81   * - Royalty recipient is not on seller's allow list
82   * - Royalty percentage is less than the value decided by the admin
83   * - Total royalty percentage is more than max royalty percentage
84   *
85   * @param _offer - the fully populated struct with offer id set to 0x0 and voided
set to false
86   * @param _offerDates - the fully populated offer dates struct
87   * @param _offerDurations - the fully populated offer durations struct
```

Example (Cont.):

```sol
88   * @param _drParameters - the id of chosen dispute resolver (can be 0) and
mutualizer address (0 for self-mutualization)
89   * @param _agentId - the id of agent
90   * @param _feeLimit - the maximum fee that seller is willing to pay per exchange
(for static offers)
91   * @param _authenticate - whether to authenticate the caller. Use false when
called from a handler that already authenticated the caller.
92   */
93  function createOfferInternal(
94      Offer memory _offer,
95      OfferDates calldata _offerDates,
96      OfferDurations calldata _offerDurations,
97      BosonTypes.DRParameters calldata _drParameters,
98      uint256 _agentId,
99      uint256 _feeLimit,
100     bool _authenticate
101 ) internal returns (uint256 offerId) {
102     address sender = _msgSender();
103
104     if (_offer.creator == OfferCreator.Seller) {
105         if (_offer.buyerId != 0) revert InvalidBuyerOfferFields();
106
107         // Validate caller is seller assistant
108         if (_authenticate) {
109             (bool isAssistant, uint256 sellerId) =
getSellerIdByAssistant(sender);
110             if (!isAssistant) {
111                 revert NotAssistant();
112             }
113             _offer.sellerId = sellerId;
114         }
115     } else if (_offer.creator == OfferCreator.Buyer) {
```

Example (Cont.):

```sol
        if (
            _offer.sellerId != 0 ||
            _offer.collectionIndex != 0 ||
            _offer.royaltyInfo.length != 1 ||
            _offer.royaltyInfo[0].recipients.length != 0 ||
            _offer.royaltyInfo[0].bps.length != 0 ||
            _drParameters.mutualizerAddress != address(0) ||
            _offer.quantityAvailable != 1 ||
            _offer.priceType != PriceType.Static
        ) {
            revert InvalidBuyerOfferFields();
        }
        if (_authenticate) _offer.buyerId = getValidBuyer(payable(sender));
    }
```

Recommendation:

We advise them to be properly documented in line with the rest of the codebase.

Alleviation (efd5d1a8f23c3bca7c25273ea4c912a367250119):

The referenced conditions have been properly documented in the function's `revert` cases, addressing this exhibit.

ProtocolBase Code Style Findings

PBE-01C: Redundant Conditional Validation

Type	Severity	Location
Gas Optimization	Informational	ProtocolBase.sol:L552

Description:

The referenced conditional will redundantly evaluate whether the `OfferCreator` enum of the `Offer` structure is a `Buyer` even though the `enum` supports two states and one of the two has already been excluded.

Example:

contracts/protocol/bases/ProtocolBase.sol

SOL

```
547 if (offer.creator == OfferCreator.Seller) {
548     // For seller-created offers, check that caller is the seller's assistant
549     (, Seller storage seller, ) = fetchSeller(offer.sellerId);
550     if (seller.assistant != _msgSender()) revert NotAssistant();
551     creatorId = offer.sellerId;
552 } else if (offer.creator == OfferCreator.Buyer) {
```

Recommendation:

We advise the referenced conditional to be set to an `else` clause, optimizing its gas cost.

Alleviation (efd5d1a8f23c3bca7c25273ea4c912a367250119):

The Boson Protocol team evaluated this exhibit and opted to retain the code as is as they consider it to be more legible and future-proof.

Finding Types

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

Input Sanitization

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

Indeterminate Code

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

Language Specific

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

Curve Specific

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

Code Style

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

Mathematical Operations

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

Logical Fault

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

Privacy Concern

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

Proof Concern

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

Severity Definition

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

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

Severity Levels

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

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

Unknown Severity

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

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

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

Informational Severity

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

Minor Severity

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

Medium Severity

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

Major Severity

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

Likelihood & Impact Assessment

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

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

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

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

Disclaimer

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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