March 20, 2024

SMART CONTRACT AUDIT REPORT

Boson Protocol
PR578 PR579

omniscia.io

info@omniscia.io

Online report: boson-protocol-pr578-pr579

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

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

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

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

omniscia.io

info@omniscia.io

PR578 PR579 Security Audit

Audit Report Revisions

Commit Hash	Date	Audit Report Hash
b29767e7be	January 4th 2024	d667f7e383
b29767e7be	January 5th 2024	6920012646
9ed49d780f	January 30th 2024	740923dcff
9ed49d780f	February 6th 2024	ee8507c082
9ed49d780f	March 20th 2024	a9c3d50fcd

Audit Overview

We were tasked with performing an audit of the Boson Protocol codebase and in particular their 578 & 579 pull requests pertaining to price discovery and royalty features introduced to the Boson Protocol.

Over the course of the audit, we identified multiple flaws in the way royalties are maintained, a crucial struct upgrade incompatibility, as well as potential exploitations of the arbitrary calls performed as part of price discovery execution flows.

We advise the Boson 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 single exhibit has been improperly dealt with. We advise the Boson Protocol team to revisit the following exhibit: `BBE-01C`

The Boson Protocol team supplied us with additional documentation as well as information in relation to their sequential commit system and proceeded to simplify their calculations by revamping their code.

As a result, we will perform a second round review of the overall system and will update the report when this is performed. The audit report must not be considered finalized until this is done.

Post-Audit Conclusion (9ed49d780f)

The Boson Protocol team re-evaluated `BBE-01C` and pointed out that it was correctly alleviated via changes to an out-of-scope contract of the audit.

We assessed these changes and confirmed that the `BBE-01C` exhibit had originally been dealt with adequately, and have since updated the exhibit's alleviation chapter to reflect that.

Additionally, a dedicated `Sequential Commit System` chapter can be observed on the side-bar that details the actions our team has taken to validate the newly introduced system within the Boson Protocol as part of this audit.

During this review, no new findings have been observed and thus the contents of the report are to be considered correct after a comprehensive security review of the Boson Protocol sequential commit system.

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	0	0	0	0
Informational	14	10	0	4
Minor	3	3	0	0
Medium	2	2	0	0
Major	2	2	0	0

During the audit, we filtered and validated a total of **5 findings utilizing static analysis** tools as well as identified a total of **16 findings during the manual review** of the codebase. We strongly recommend that any minor severity or higher findings are dealt with promptly prior to the project's launch as they can introduce potential misbehaviours of the system as well as exploits.

Scope

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

Target

- Repository: **https://github.com/bosonprotocol/boson-protocol-contracts**
- Commit: b29767e7be7b14ff87e69fb2709b4324956b93df
- Language: Solidity
- Network: Polygon,Matic,Ethereum
- Revisions: **b29767e7be**, **9ed49d780f**, **9ed49d780f**

Contracts Assessed

File	Total Finding(s)
contracts/protocol/bases/BuyerBase.sol (BBE)	1
contracts/domain/BosonTypes.sol (BTS)	1
contracts/domain/BosonErrors.sol (BES)	1
contracts/protocol/clients/voucher/BosonVoucher.sol (BVR)	0
contracts/domain/BosonConstants.sol (BCS)	0
contracts/protocol/bases/BeaconClientBase.sol (BCB)	0
contracts/protocol/facets/ConfigHandlerFacet.sol (CHF)	0
contracts/protocol/bases/DisputeBase.sol (DBE)	0
contracts/protocol/facets/ExchangeHandlerFacet.sol (EHF)	0
contracts/protocol/libs/FundsLib.sol (FLB)	5

contracts/protocol/facets/FundsHandlerFacet.sol (FHF)	0
contracts/protocol/bases/OfferBase.sol (OBE)	0
contracts/protocol/facets/OfferHandlerFacet.sol (OHF)	1
contracts/protocol/facets/OrchestrationHandlerFacet1.sol (OH1)	0
contracts/protocol/libs/ProtocolLib.sol (PLB)	0
contracts/protocol/bases/ProtocolBase.sol (PBE)	0
contracts/protocol/bases/PriceDiscoveryBase.sol (PDB)	4
contracts/protocol/facets/PriceDiscoveryHandlerFacet.sol (PDH)	1
contracts/protocol/bases/SellerBase.sol (SBE)	1
contracts/protocol/facets/SellerHandlerFacet.sol (SHF)	3
contracts/protocol/facets/SequentialCommitHandlerFacet.sol (SCH)	3

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 conatined in dependencies as well as example contracts and can thus be safely ignored.

The `pragma` version has been locked to `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 **131 potential issues** within the codebase of which **107 were ruled out to be false positives** or negligible findings.

The remaining **24 issues** were validated and grouped and formalized into the **5 exhibits** that follow:

ID	Severity	Addressed	Title
FLB-01S	Informational	Acknowledged	Illegible Numeric Value Representation
PDB-01S	Informational	Yes	Unutilized Contract Member
PDB-02S	Minor	Yes	Inexistent Sanitization of Input Address
PDH-01S	Informational	Acknowledged	Illegible Numeric Value Representation
SCH-01S	Informational	Acknowledged	Illegible Numeric Value Representation

Manual Review

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

As the project at hand implements royalties and NFT price discovery integrations, intricate care was put into ensuring that the **flow of funds within the system conforms to the specifications and restrictions** laid forth within the protocol's specification and that **the execution flows of the contract cannot be compromised by external calls**.

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; we urge the Boson Protocol team to promptly evaluate and remediate them.

Additionally, the system was investigated for any other commonly present attack vectors such as re-entrancy attacks, mathematical truncations, logical flaws and **ERC / EIP** standard inconsistencies. The documentation of the project was satisfactory to a certain extent, however, we strongly recommend it to be expanded at certain complex points such as the sequential commit price calculations.

A total of **16 findings** were identified over the course of the manual review of which **7 findings** concerned the behaviour and security of the system. The non-security related findings, such as optimizations, are included in the separate **Code Style** chapter.

The finding table below enumerates all these security / behavioural findings:

ID	Severity	Addressed	Title
BTS-01M	Major	Nullified	Incorrect Location of Introduced Variable
FLB-01M	Informational	Yes	Potentially Incorrect Dispute Case Handling
FLB-02M	Minor	Yes	Potential Gas Bombing Attack Vector
PDB-01M	Major	Yes	Arbitrary External Contract Calls
SBE-01M	Medium	Yes	Incorrect Default Royalty Recipient Initialization

SHF-01M	Medium	Yes	Inexistent Erasure of Recipient Index
SCH-01M	Minor	Yes	Incorrect Specification of Error

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
BES-01C	Informational	Acknowledged	Lingering TODO Comments
BBE-01C	Informational	Yes	Inefficient Creation of Buyer
FLB-01C	Informational	Yes	Redundant Self-Assignment
FLB-02C	Informational	Yes	Repetitive Value Literal
OHF-01C	Informational	Yes	Redundant Application of Security Modifier
PDB-01C	Informational	Nullified	Inexistent Error Message
SHF-01C	Informational	Yes	Inefficient `mapping` Lookups
SHF-02C	Informational	Yes	Non-Uniform Royalty Recipient ID Definition
SCH-01C	Informational	Yes	Ineffectual Usage of Safe Arithmetics

Sequential Commit System

A significant portion of the audit effort was invested in properly validating that the newly introduced sequential commit system in the Boson Protocol behaves according to its specification. This audit effort was mainly performed on commit hash `9ed49d780f04b2f168001f72d6c09f5fc94818b7` after the Boson Protocol team supplemented us with additional information about the system.

We relied on the Boson Protocol Improvement Proposal **BPIP-7** as well as the accompanying **PDF document** to validate whether the system behaves correctly.

Over the course of our audit, we identified several discrepancies between the formulae within the PDF and the actual implementations in the codebase. We believe that some are the result of outdated definitions (i.e. lack of an escalation fee) whilst others have directly translated to vulnerabilities within the audit report.

To note, the model we validated the codebase against is the capital-time optimized model which properly corresponds to the logic implemented within `FundsLib::releaseFundsToIntermediateSellers`.

System Principles

A total of 12 system principles can be observed in the aforementioned PDF which were treated as invariants over the course of this audit round.

During our evaluation of the principles, we confirmed that principles N1 and N3 are not properly upheld in the system. Specifically, the `FundsLib` payout mechanism will use the `BosonTypes::Offer` price instead of the final entry in the `exchangeCosts` array which represents the initial price of the offer.

This is further confirmed by the fact that the input argument of `FundsLib::releaseFundsToIntermediateSellers` is specified as `_initialPrice` yet is treated as the final price in the `buyerPayoff` assignments performed in `FundsLib::releaseFunds`.

We advise the Boson Protocol team to rectify this vulnerability by properly distinguishing the initial price and the final price of a sale within the `FundsLib::releaseFunds` function.

Interim Cash Flows

One of the items lacking in the system's documentation is a cash flow function which should depict the amount of funds each interim buyer must escrow and each seller acquires in the capital-time optimized model.

The escrowed amount per sale is tracked in the `SequentialCommitHandlerFacet::sequentialCommitToOffer` function and is as follows:

$$escrowedAmount = p_{i+1} - min\{p_i, p_{i+1} - r_{i+1} - pf_{i+1}\}$$

In the above equation:

The immediately released payout per sale is equal to the sale price minus the amount meant to be escrowed:

$$immediatePayout = p_{i+1} - escrowedAmount$$

We validated that the amounts above conform with the sequential commit specification and particularly with principle N6 which we have treated as an invariant in the system.

A problem we identified in the interim cash flows is how in an `Ask` based price discovery sequential commit using native funds (i.e. `exchangeToken == address(0)`), the system will attempt to fetch the additionally escrowed fees using the wrapped variant of the native token.

This approach is invalid as the price discovery system will supply native funds to the call it makes to acquire the NFT, meaning that the seller would most likely have received native funds as well.

This represents a capital inefficient model whereby the seller would need to have wrapped assets at rest that are unrelated to the sale proceeds and in most cases this will lead to a transaction failure.

To avoid this, we advise a similar approach to the `PriceDiscoveryBase::fulfilBidOrder` function to be taken using the wrapped native asset when performing the ask order's fulfilment.

On a final note, we observed that the fulfilment of a bid order is detached from the input `_buyer`. Specifically, the address of the input `_buyer` is ignored and whoever acquires the NFT after a price discovery sale is automatically tracked as the new buyer.

While slightly non-standard (i.e. the `_buyer` argument should not be validated as non-zero in bid orders, for example), we consider this behavior correct and did not identify a vulnerability arising from it.

Price Discovery Executions

The sequential commit system integrates with the price discovery system of the Boson Protocol to rely on non-protocol contracts to acquire the Boson Protocol NFT using an external exchange.

These price discovery mechanisms rely on pre- and post-execution measurements of the contract's balance.

Re-entrancy attacks would cause these measurements to result in major vulnerabilities as a re-entry would cause the post-balance measurement of the original call to take into account new inflows performed in the inner re-entrant call(s).

The system imposes re-entrancy guards across the board, and it is imperative that the re-entrancy guards of functions integrating with the `PriceDiscoveryBase` contract are **shared and properly enforced**.

In the current implementation, the functions that integrate with the `PriceDiscoveryBase` are both protected properly and are the `PriceDiscoveryHandlerFacet::commitToPriceDiscoveryOffer` and `SequentialCommitHandlerFacet::sequentialCommitToOffer`.

Final State Cash Flows

As per an earlier chapter, we identified a discrepancy in the final state cash flows whereby the final buyer would receive an incorrect amount of funds in case of a non-happy path.

Beyond the final buyer payout discrepancy, we did not identify any other flaw in how the cash flows at the end of an offer are handled.

One **implied restriction of the current system** is the fact that royalty entries should remain immutable and the royalty structure of an offer should be an append-only structure.

This is due to the fact that the intermediate buyer payouts rely on index-based access of royalty entries whose contents are expected to match those that were fetched when the intermediate buyer made their purchase.

It is imperative that this trait is upheld as otherwise offer fulfilments may fail to occur or may result in lost funds.

Actor Incentive Misalignments

The current sequential commit system contains a few processes that misalign the incentives of buyers as well as the original seller.

In detail, the following final states have a misaligned incentive for the original seller:

In the above scenarios, a seller is incentivized to buy their original sale on the open market using the sequential commit system f.e. by using flash-loans.

In doing so, they will result in a net-neutral state (i.e. `0`) for themselves as they would acquire the seller deposit themselves. This type of incentive cannot be prevented as a seller can use a secondary address that they own to perform the "attack".

Another incentive misalignment present in the current system is the fact that intermediate sellers who sold at a loss are incentivized to have the overall offer undergo a dispute.

Depending on the intermediate steps in the sale, an interim buyer can collaborate with the final buyer and original seller to maliciously request a dispute.

Taking the above manipulation vector further, the interim buyer can simultaneously be the original seller and final buyer as a single actor.

This permits an individual to craft a sequence of commits for an offer and solely accept the sequence if they have turned a profit while freely performing market manipulation as they are able to retroactively cancel all their at-loss sales.

The above incentive misalignments should be taken into account as they can result in a system that is ultimately exploited at the expense of normal Boson Protocol users.

Outdated Formulae

As a concluding note, the formulae within the PDF lack the definition of protocol fees, buyer cancellation penalties, agent fees, and buyer escalation deposits.

Additionally, some of the formulae improperly depict the seller deposit which is also refunded in the happy-path execution flows. This results in an incorrect depiction of the `Resolved / Decided` state for the `Seller` as well as other inconsistencies in these calculations.

While we have validated that they are properly handled in the system, we advise their inclusion in the PDF to ensure that it depicts the latest state of the Boson Protocol system.

Post-Audit Conclusion (6fa4d9b5c0)

The Boson Protocol team provided us with a follow-up commit to evaluate fixes that were carried out as a result of this report as well as an internal review of the contracts by the Boson Protocol team.

`FundsLib.sol`

The vulnerability identified in the `System Principles` chapter has been rectified by distinguishing between the initial `offerPrice` and the final `lastPrice`, reimbursing the seller and last buyer respectively as expected.

The code was refactored to accommodate for the above fact as well as utilize a `FundsLib::applyPercent` utility function that greatly simplified the code and increased its legibility.

Finally, the `protocolFee` calculations within `FundsLib::releaseFundsToIntermediateSellers` incorporate the `effectivePriceMultiplier` reduction directly instead of at the end presumably to accommodate for truncations at each percentage calculation rather than at the end.

`PriceDiscoveryBase.sol`, `SequentialCommitHandlerFacet.sol`, and `PriceDiscoveryHandlerFacet.sol`

The capital inefficiency flaw we identified in the `Interim Cash Flows` chapter has been resolved by ensuring that the wrapped variant of the native token is utilized in the `PriceDiscoveryBase::fulfilAskOrder` function flow.

An inward transfer of the `actualPrice` has been introduced to the `PriceDiscoveryBase` ask-based fulfilment flow and the previously present inward flows in the `SequentialCommitHandlerFacet` and `PriceDiscoveryHandlerFacet` have been removed.

The above changes all related to the original capital inefficiency we identified and have thus properly addressed it.

Supplemental Changes

The Boson Protocol team introduced some supplemental changes to the above that introduce two new pausable regions in the Boson Protocol system as well as the introduction of two additional validations for offer creations; a `Discovery` price type should have an offer price of `0` and a price in the sequential commit flow should always exceed the buyer's cancellation penalty.

The above changes were minimal and have been validated to fulfil their purpose with no new vulnerabilities arising as a result of their introduction.

FundsLib Static Analysis Findings

FLB-01S: Illegible Numeric Value Representation

Type	Severity	Location
Code Style	Informational	FundsLib.sol:L260, L275, L281, L321, L323, L344, L565, L568, L584

Description:

The linked representation of a numeric literal is sub-optimally represented decreasing the legibility of the codebase.

Example:

contracts/protocol/libs/FundsLib.sol

```sol
effectivePriceMultiplier = 10000;
```

Recommendation:

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

Alleviation (9ed49d780f04b2f168001f72d6c09f5fc94818b7):

The Boson Protocol team has specified that they do not use the underscore-based style to represent value literals and as such wish to retain the current representation.

As such, we consider this exhibit properly acknowledged.

PriceDiscoveryBase Static Analysis Findings

PDB-01S: Unutilized Contract Member

Type	Severity	Location
Gas Optimization	Informational	PriceDiscoveryBase.sol:L24

Description:

The `EXCHANGE_ID_2_2_0` member of the `PriceDiscoveryBase` contract remains unutilized.

Example:

contracts/protocol/bases/PriceDiscoveryBase.sol

SOL

```sol
24 uint256 private immutable EXCHANGE_ID_2_2_0; // solhint-disable-line
25
26 /**
27  * @notice
28  * For offers with native exchange token, it is expected the the price discovery contracts will
29  * operate with wrapped native token. Set the address of the wrapped native token in the constructor.
30  *
31  * After v2.2.0, token ids are derived from offerId and exchangeId.
32  * EXCHANGE_ID_2_2_0 is the first exchange id to use for 2.2.0.
33  * Set EXCHANGE_ID_2_2_0 in the constructor.
```

Example (Cont.):

SOL

```
 *
 * @param _wNative - the address of the wrapped native token
 * @param _firstExchangeId2_2_0 - the first exchange id to use for 2.2.0
 */
//solhint-disable-next-line
constructor(address _wNative, uint256 _firstExchangeId2_2_0) {
    wNative = IWrappedNative(_wNative);
    EXCHANGE_ID_2_2_0 = _firstExchangeId2_2_0;
}
```

Recommendation:

We advise it to either be utilized properly or omitted from the codebase, either of which we consider an adequate remediation to this exhibit.

Alleviation (9ed49d780f04b2f168001f72d6c09f5fc94818b7):

The unutilized contract member has been safely omitted.

PDB-02S: Inexistent Sanitization of Input Address

Type	Severity	Location
Input Sanitization	Minor	PriceDiscoveryBase.sol:L39-L42

Description:

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

Impact:

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

Example:

contracts/protocol/bases/PriceDiscoveryBase.sol

SOL

```
39 constructor(address _wNative, uint256 _firstExchangeId2_2_0) {
40     wNative = IWrappedNative(_wNative);
41     EXCHANGE_ID_2_2_0 = _firstExchangeId2_2_0;
42 }
```

Recommendation:

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

Alleviation (9ed49d780f04b2f168001f72d6c09f5fc94818b7):

The input `_wNative` address argument of the **`PriceDiscoveryBase::constructor`** function is adequately sanitized as non-zero in the latest in-scope revision of the codebase, addressing this exhibit.

PriceDiscoveryHandlerFacet Static Analysis Findings

PDH-01S: Illegible Numeric Value Representation

Type	Severity	Location
Code Style	Informational	PriceDiscoveryHandlerFacet.sol:L130

Description:

The linked representation of a numeric literal is sub-optimally represented decreasing the legibility of the codebase.

Example:

contracts/protocol/facets/PriceDiscoveryHandlerFacet.sol

SOL

```sol
130 uint256 royaltyAmount = (getTotalRoyaltyPercentage(royaltyInfo.bps) * actualPrice) / 10000;
```

Recommendation:

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

Alleviation (9ed49d780f04b2f168001f72d6c09f5fc94818b7):

The Boson Protocol team has specified that they do not use the underscore-based style to represent value literals and as such wish to retain the current representation.

As such, we consider this exhibit properly acknowledged.

SequentialCommitHandlerFacet Static Analysis Findings

SCH-01S: Illegible Numeric Value Representation

Type	Severity	Location
Code Style	Informational	SequentialCommitHandlerFacet.sol:L134

Description:

The linked representation of a numeric literal is sub-optimally represented decreasing the legibility of the codebase.

Example:

contracts/protocol/facets/SequentialCommitHandlerFacet.sol

SOL

```
10000;
```

Recommendation:

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

Alleviation (9ed49d780f04b2f168001f72d6c09f5fc94818b7):

The Boson Protocol team has specified that they do not use the underscore-based style to represent value literals and as such wish to retain the current representation.

As such, we consider this exhibit properly acknowledged.

BosonTypes Manual Review Findings

BTS-01M: Incorrect Location of Introduced Variable

Type	Severity	Location
Language Specific	Major	BosonTypes.sol:L156

Description:

The `Offer` struct within `BosonTypes` had two new variables introduced, one of which has been introduced in between the structure's entries. This is invalid, as an upgrade of the Boson Protocol will shift all ensuing entries by one storage slot and will cause `priceType` to be represented by the `metadataUri` entry.

Impact:

If an upgrade of the Boson Protocol system is performed in its current state, the `priceType` of all previous orders will incorrectly be represented by the `metadataUri` entry and specifically its length. Likewise, all ensuing entries (`metadataUri`, `metadataHash`, `voided`, `collectionIndex`) will be shifted by one word corrupting their data and causing f.e. previously `voided` offers to be "enabled" if they had a `collectionIndex` with the first bit set to `1`.

Example:

contracts/domain/BosonTypes.sol

```sol
struct Offer {
    uint256 id;
    uint256 sellerId;
    uint256 price;
    uint256 sellerDeposit;
    uint256 buyerCancelPenalty;
    uint256 quantityAvailable;
    address exchangeToken;
    PriceType priceType;
    string metadataUri;
```

Example (Cont.):

```sol
    string metadataHash;
    bool voided;
    uint256 collectionIndex;
    RoyaltyInfo[] royaltyInfo;
}
```

Recommendation:

We advise the `priceType` entry to be relocated at the end of the `Offer` structure, ensuring that an upgrade of the Boson Protocol will correctly "initialize" the `priceType` to its default value of `PriceType::Static`.

Alleviation (9ed49d780f04b2f168001f72d6c09f5fc94818b7):

The `PriceType` member represents an `enum` rather than a `struct` and since Solidity `0.8.0` such declarations are guaranteed to fit within `1` byte, permitting them to be tight-packed with the preceding `address` member.

While the entries are indeed not shifted, we would advise against introduction of variables in between declarations to avoid latent compilation bugs as it is not standard practice. In any case, **we consider the original exhibit invalid and thus nullify this exhibit**.

FundsLib Manual Review Findings

FLB-01M: Potentially Incorrect Dispute Case Handling

Type	Severity	Location
Logical Fault	Informational	FundsLib.sol:L176, L280

Description:

A `DisputeState` of `Escalated` will cause the execution path for `Resolved` / `Decided` to be executed which may be incorrect.

Impact:

The severity of this exhibit will be adjusted according to the action the Boson Protocol team takes to rectify it.

Example:

contracts/protocol/libs/FundsLib.sol

SOL

```
if (disputeState == BosonTypes.DisputeState.Retracted) {
    // RETRACTED - same as "COMPLETED"
    protocolFee = offerFee.protocolFee;
    agentFee = offerFee.agentFee;
    // buyerPayoff is 0
    sellerPayoff = price + sellerDeposit - protocolFee - agentFee + buyerEscalationDeposit;
} else if (disputeState == BosonTypes.DisputeState.Refused) {
    // REFUSED
    sellerPayoff = sellerDeposit;
    buyerPayoff = price + buyerEscalationDeposit;
```

Example (Cont.):

SOL

```sol
} else {
    // RESOLVED or DECIDED
    uint256 pot = price + sellerDeposit + buyerEscalationDeposit;
    buyerPayoff = (pot * dispute.buyerPercent) / 10000;
    sellerPayoff = pot - buyerPayoff;
}
```

Recommendation:

We advise the Boson Protocol team to evaluate this and either update the documentation of the code or introduce a new `if-revert` check that prevents an `Escalated` dispute from executing the `Resolved` / `Decided` execution path, the latter of which we consider the correct rectification for this exhibit.

Alleviation (9ed49d780f04b2f168001f72d6c09f5fc94818b7):

The Boson Protocol team investigated the described execution path and denoted that it is unreachable due to the way escalated cases will never execute the referenced code.

Regardless of this fact, the Boson Protocol team has proceeded with adding an additional sanity check to the `DisputeHandlerFacet::finalizeDispute` function to ensure that the `_targetState` passed in cannot be `DisputeState::Escalated` or `DisputeState::Resolving` further reinforcing the fact that the code of **`FundsLib::releaseFunds`** will not execute for the described dispute states.

FLB-02M: Potential Gas Bombing Attack Vector

Type	Severity	Location
Logical Fault	Minor	FundsLib.sol:L574-L575

Description:

The `FundsLib::distributeRoyalties` function that is invoked as part of a `FundsLib::releaseFundsToIntermediateSellers` function execution is susceptible to gas bombing whereby a malicious recipient can consume the gas of the transaction and thus cause it to fail.

Impact:

The likelihood of a royalty recipient misbehaving on receipt of native funds is low rendering this exhibit to be of minor severity.

Example:

contracts/protocol/libs/FundsLib.sol

SOL

```
/**
 * @notice Distributes the royalties to external recipients and seller's treasury.
 *
 * @param _offer - storage pointer to the offer
 * @param _royaltyInfoIndex - index of the royalty info (reffers to offer.royaltyInfo array)
 * @param _price - price in the sequential commit
 * @param _escrowedRoyaltyAmount - amount of royalties that were escrowed
 * @param _effectivePriceMultiplier - multiplier for the price, depending on the state of the exchange
 */
function distributeRoyalties(
```

Example (Cont.):

```sol
    BosonTypes.Offer storage _offer,
    uint256 _royaltyInfoIndex,
    uint256 _price,
    uint256 _escrowedRoyaltyAmount,
    uint256 _effectivePriceMultiplier
) internal returns (uint256 sellerRoyalties) {
    address exchangeToken = _offer.exchangeToken;
    BosonTypes.RoyaltyInfo storage _royaltyInfo = _offer.royaltyInfo[_royaltyInfoIndex];
    uint256 len = _royaltyInfo.recipients.length;
    uint256 totalAmount;
    uint256 effectivePrice = (_price * _effectivePriceMultiplier) / 10000;
    for (uint256 i = 0; i < len; ) {
        address payable recipient = _royaltyInfo.recipients[i];
        uint256 amount = (_royaltyInfo.bps[i] * effectivePrice) / 10000;
        totalAmount += amount;
        if (recipient == address(0)) {
            // goes to seller's treasury
            sellerRoyalties = amount;
        } else {
            // try to transfer the funds. Or better make it available to withdraw?
            FundsLib.transferFundsFromProtocol(exchangeToken, recipient, amount);
        }

        unchecked {
            i++;
        }
    }

```

Example (Cont.):

```sol
    // if there is a remainder due to rounding, it goes to the seller's treasury
    sellerRoyalties += (_effectivePriceMultiplier * _escrowedRoyaltyAmount) / 10000 - totalAmount;
}
```

Recommendation:

We advise the code to instead credit the `recipient` with the relevant amount instead of directly transferring to it, ensuring misbehaving royalty recipients do not cause payment releases to fail if they involve native funds.

Alleviation (9ed49d780f04b2f168001f72d6c09f5fc94818b7):

The royalty distribution pattern has adopted a credit system as we advised, utilizing the existing checks-and-balances system of Boson to credit the royalty recipients with the amount of funds they are due.

As a result of this change, we consider the exhibit fully alleviated as an external call is no longer performed.

PriceDiscoveryBase Manual Review Findings

PDB-01M: Arbitrary External Contract Calls

Type	Severity	Location
Logical Fault	Major	PriceDiscoveryBase.sol:L132, L210, L275

Description:

The `PriceDiscoveryBase` contract permits arbitrary contract calls to be made with arbitrary payloads, opening up multiple attack vectors that can ultimately lead to a compromise of protocol funds.

As an example, any Boson voucher that is owned by the protocol itself can be exploited and acquired by an arbitrary party for free via the **`PriceDiscoveryBase::handleWrapper`** execution path by setting the `_priceDiscovery.priceDiscoveryContract` as the Boson Protocol Diamond and the `_priceDiscovery.priceDiscoveryData` payload as a `transferFrom` or `safeTransferFrom` **EIP-721** function.

Impact:

Arbitrary calls to arbitrary contracts should not be allowed by any contract as they pose a significant security risk and can be exploited to siphon funds from the protocol. In the case of the Boson Protocol, this may be possible if an **EIP-20** asset supports a transfer-and-call flow (i.e. **ERC-677**) or is an **EIP-721** / **EIP-1155** asset.

Example:

contracts/protocol/bases/PriceDiscoveryBase.sol

```sol
252 function handleWrapper(
253     uint256 _tokenId,
254     address _exchangeToken,
255     PriceDiscovery calldata _priceDiscovery,
256     IBosonVoucher _bosonVoucher
257 ) internal returns (uint256 actualPrice) {
258     if (_tokenId == 0) revert TokenIdMandatory();
259 
260     // If price discovery contract does not own the voucher, it cannot be classified as a wrapper
261     address owner = _bosonVoucher.ownerOf(_tokenId);
```

Example (Cont.):

```sol
    if (owner != _priceDiscovery.priceDiscoveryContract) revert NotVoucherHolder();

    // Check balance before calling wrapper
    bool isNative = _exchangeToken == address(0);
    if (isNative) _exchangeToken = address(wNative);
    uint256 protocolBalanceBefore = getBalance(_exchangeToken, address(this));

    // Track native balance just in case if seller sends some native currency.
    // All native currency is forwarded to the wrapper, which should not return any back.
    // If it does, we revert later in the code.
    uint256 protocolNativeBalanceBefore = getBalance(address(0), address(this)) - msg.value;

    // Call the price discovery contract
    _priceDiscovery.priceDiscoveryContract.functionCallWithValue(_priceDiscovery.priceDiscoveryData, msg.value);

    // Check the native balance and revert if there is a surplus
    uint256 protocolNativeBalanceAfter = getBalance(address(0), address(this));
    require(protocolNativeBalanceAfter == protocolNativeBalanceBefore);

    // Check balance after the price discovery call
    uint256 protocolBalanceAfter = getBalance(_exchangeToken, address(this));

    // Verify that actual price is within the expected range
    if (protocolBalanceAfter < protocolBalanceBefore) revert NegativePriceNotAllowed();
    actualPrice = protocolBalanceAfter - protocolBalanceBefore;

    // when working with wrappers, price is already known, so the caller should set it exactly
    // If protocol receive more than expected, it does not return the surplus to the caller
```

Example (Cont.):

```sol
    if (actualPrice != _priceDiscovery.price) revert PriceTooLow();

    // Verify that token id provided by caller matches the token id that the price discovery contract has sent to buyer
    getAndVerifyTokenId(_tokenId);
}
```

Recommendation:

We advise the Boson Protocol to maintain a set of whitelisted `priceDiscoveryContract` entries, sanitizing the `_priceDiscovery.priceDiscoveryContract` entries. As an additional security measure, we advise the signatures permitted to be whitelisted as well by validating the first four bytes of the `_priceDiscovery::priceDiscoveryData` payload.

Alleviation (9ed49d780f04b2f168001f72d6c09f5fc94818b7):

The Boson Protocol team evaluated this exhibit and has opted to implement an alternative alleviation, utilizing a secondary contract for the arbitrary calls.

As the secondary contract is never meant to retain funds at rest, we consider this approach correct. The actual implementation of the overall flow will be evaluated at a later point to ensure that the decoupling of logic has been performed safely.

SellerBase Manual Review Findings

SBE-01M: Incorrect Default Royalty Recipient Initialization

Type	Severity	Location
Logical Fault	Medium	SellerBase.sol:L99

Description:

The `SellerBase::createSellerInternal` function will initialize the default royalty recipient of the seller (i.e. `address(0)`) without setting its `royaltyRecipientIndexBySellerAndRecipient` entry. As a result, the default recipient can be re-added via the `SellerHandlerFacet::addRoyaltyRecipients` and `SellerHandlerFacet::updateRoyaltyRecipients` functions incorrectly, causing the overall royalty system of the contract to misbehave.

Impact:

It is presently possible for the `treasury` to exist twice as a royalty recipient for a particular seller which we consider a major invariant breach of the protocol.

Example:

contracts/protocol/bases/SellerBase.sol

```sol
98  RoyaltyRecipient[] storage royaltyRecipients = lookups.royaltyRecipientsBySeller[sellerId];
99  RoyaltyRecipient storage defaultRoyaltyRecipient = royaltyRecipients.push();
100 // We don't store the defaultRoyaltyRecipient.wallet, since it's always the trasury
101 // We don't store the defaultRoyaltyRecipient.externalId, since the default recipient is always the treasury
102 defaultRoyaltyRecipient.minRoyaltyPercentage = _voucherInitValues.royaltyPercentage;
```

Recommendation:

We advise the `royaltyRecipientIndexBySellerAndRecipient` entry of the `address(0)` to be properly maintained, preventing it from being re-added and thus ensuring that the `treasury` recipient exists only once.

Alleviation (9ed49d780f04b2f168001f72d6c09f5fc94818b7):

The **`SellerHandlerFacet::updateRoyaltyRecipients`** and **`SellerHandlerFacet::addRoyaltyRecipients`** functions were updated to not allow the zero-address to be re-added or have its index mutated, effectively ensuring it is "immutable" at the initial position (`0`) of the array.

As a result, this exhibit is alleviated given that the `defaultRoyaltyRecipient` is correctly initialized and maintained by the overall `SellerBase` system.

SellerHandlerFacet Manual Review Findings

SHF-01M: Inexistent Erasure of Recipient Index

Type	Severity	Location
Logical Fault	Medium	SellerHandlerFacet.sol:L170-L172

Description:

The code of the `SellerHandlerFacet::updateSeller` function will incorrectly maintain the `royaltyRecipientIndexBySellerAndRecipient` entry as it will not overwrite it if the `royaltyRecipientId` matches the `lastRoyaltyRecipientsId`. This means that the new treasury will have a non-zero `royaltyRecipientIndexBySellerAndRecipient` that will affect future invocations of the `SellerHandlerFacet::updateRoyaltyRecipients` and the `SellerHandlerFacet::addRoyaltyRecipients` functions if the treasury is changed again.

Impact:

Presently, a royalty recipient that is added as a `treasury` and is then removed will be impossible to re-add to the royalty system. We consider this to be an invalid trait of the system that has a decent likelihood of manifesting thus rendering this exhibit to be of medium severity.

Example:

contracts/protocol/facets/SellerHandlerFacet.sol

SOL

```solidity
if (_seller.treasury != seller.treasury) {
    if (_seller.treasury == address(0)) revert InvalidAddress();

    // Check if new treasury is already a royalty recipient
    mapping(address => uint256) storage royaltyRecipientIndexBySellerAndRecipient = lookups
        .royaltyRecipientIndexBySellerAndRecipient[_seller.id];
    uint256 royaltyRecipientId = royaltyRecipientIndexBySellerAndRecipient[_seller.treasury];

    if (royaltyRecipientId != 0) {
        RoyaltyRecipient[] storage royaltyRecipients = lookups.royaltyRecipientsBySeller[_seller.id];
```

Example (Cont.):

SOL

```sol

        // If the new treasury is already a royalty recipient, remove it
        royaltyRecipientId--; // royaltyRecipientId is 1-based, so we need to decrement it to get the index
        uint256 lastRoyaltyRecipientsId = royaltyRecipients.length - 1;
        if (royaltyRecipientId != lastRoyaltyRecipientsId) {
            royaltyRecipients[royaltyRecipientId] = royaltyRecipients[lastRoyaltyRecipientsId];
            royaltyRecipientIndexBySellerAndRecipient[royaltyRecipients[royaltyRecipientId].wallet] =
                royaltyRecipientId +
                1;
        }
        royaltyRecipients.pop();
    }

    // Update treasury
    seller.treasury = _seller.treasury;

    updateApplied = true;

    emit RoyaltyRecipientsChanged(_seller.id, fetchRoyaltyRecipients(_seller.id), msgSender());
}
```

Recommendation:

We advise the `royaltyRecipientIndexBySellerAndRecipient` entry of the **previous** treasury to be set to `0`, ensuring that the treasury can be re-added as a royalty recipient if it is changed in the future.

Alleviation (9ed49d780f04b2f168001f72d6c09f5fc94818b7):

The code was updated per our recommendation, deleting the previous `royaltyRecipientIndexBySellerAndRecipient` entry and thus ensuring that they can be re-added if need be.

SequentialCommitHandlerFacet Manual Review Findings

SCH-01M: Incorrect Specification of Error

Type	Severity	Location
Standard Conformity	Minor	SequentialCommitHandlerFacet.sol:L68

Description:

The `SequentialCommitHandlerFacet::sequentialCommitToOffer` function specifies that the code should fail if the buyer is unable to receive the Boson voucher NFT, however, this is not the case as the `PriceDiscoveryBase::fulfilAskOrder` function will simply invoke the `ERC721Upgradeable::transferFrom` function instead of the `ERC721Upgradeable::safeTransferFrom` variant.

Impact:

The severity of this exhibit will be adjusted accordingly once the Boson Protocol team remediates it.

Example:

contracts/protocol/facets/SequentialCommitHandlerFacet.sol

SOL

```
 *   - Transfer of voucher to the buyer fails for some reasong (e.g. buyer is contract that doesn't accept voucher)
```

Recommendation:

We advise either the documentation or the code to be updated, either of which we consider an adequate resolution to this exhibit.

Alleviation (9ed49d780f04b2f168001f72d6c09f5fc94818b7):

The `PriceDiscoveryBase::fulfilAskOrder` function was updated to perform an `ERC721Upgradeable::safeTransferFrom` interaction upholding the security detailed in the function's documentation and thus alleviating this exhibit.

BosonErrors Code Style Findings

BES-01C: Lingering TODO Comments

Type	Severity	Location
Code Style	Informational	BosonErrors.sol:L9, L25, L27, L31, L40, L234, L262

Description:

The referenced comments represent `TODO` statements that have yet to be resolved in the codebase.

Example:

contracts/domain/BosonErrors.sol

SOL

```
// Whenever a region is paused, and a method from that region is called
error RegionPaused(); //ToDo consider adding the region to the error message
```

Recommendation:

We advise them to be resolved, especially the ones referencing input arguments to `error` declarations as they would be inconsequential gas-wise and would greatly enhance off-chain debugging capabilities.

Alleviation (9ed49d780f04b2f168001f72d6c09f5fc94818b7):

The Boson Protocol team evaluated the `TODO` comments and due to their non-critical nature will re-evaluate them at a later point, rendering this exhibit acknowledged.

BuyerBase Code Style Findings

BBE-01C: Inefficient Creation of Buyer

Type	Severity	Location
Gas Optimization	Informational	BuyerBase.sol:L90

Description:

The `BuyerBase::getValidBuyer` function will automatically create a buyer account if one does not exist for the `_buyer`, however, this is done so inefficiently as the `active` entry of the `Buyer` struct as well as the `buyerIdByWallet` mapping will be inefficiently evaluated.

Example:

contracts/protocol/bases/BuyerBase.sol

```sol
/**
 * @notice Creates a Buyer.
 *
 * Emits a BuyerCreated event if successful.
 *
 * Reverts if:
 * - Wallet address is zero address
 * - Active is not true
 * - Wallet address is not unique to this buyer
 *
```

Example (Cont.):

```sol
 * @param _buyer - the fully populated struct with buyer id set to 0x0
 */
function createBuyerInternal(Buyer memory _buyer) internal {
    //Check for zero address
    if (_buyer.wallet == address(0)) revert InvalidAddress();

    //Check active is not set to false
    if (!_buyer.active) revert MustBeActive();

    // Get the next account id and increment the counter
    uint256 buyerId = protocolCounters().nextAccountId++;

    //check that the wallet address is unique to one buyer id
    if (protocolLookups().buyerIdByWallet[_buyer.wallet] != 0) revert BuyerAddressMustBeUnique();

    _buyer.id = buyerId;
    storeBuyer(_buyer);

    //Notify watchers of state change
    emit BuyerCreated(_buyer.id, _buyer, msgSender());
}

/**
 * @notice Stores buyer struct in storage.
 *
 * @param _buyer - the fully populated struct with buyer id set
 */
function storeBuyer(Buyer memory _buyer) internal {
```

Example (Cont.):

SOL

```solidity
    // Get storage location for buyer
    (, Buyer storage buyer) = fetchBuyer(_buyer.id);

    // Set buyer props individually since memory structs can't be copied to storage
    buyer.id = _buyer.id;
    buyer.wallet = _buyer.wallet;
    buyer.active = _buyer.active;

    //Map the buyer's wallet address to the buyerId.
    protocolLookups().buyerIdByWallet[_buyer.wallet] = _buyer.id;
}

/**
 * @notice Checks if buyer exists for buyer address. If not, account is created for buyer address.
 *
 * Reverts if buyer exists but is inactive.
 *
 * @param _buyer - the buyer address to check
 * @return buyerId - the buyer id
 */
function getValidBuyer(address payable _buyer) internal returns (uint256 buyerId) {
    // Find or create the account associated with the specified buyer address
    bool exists;
    (exists, buyerId) = getBuyerIdByWallet(_buyer);

    if (exists) {
        // Fetch the existing buyer account
        (, Buyer storage buyer) = fetchBuyer(buyerId);
```

Example (Cont.):

```sol

        // Make sure buyer account is active
        if (!buyer.active) revert MustBeActive();
    } else {
        // Create the buyer account
        Buyer memory newBuyer;
        newBuyer.wallet = _buyer;
        newBuyer.active = true;

        createBuyerInternal(newBuyer);
        buyerId = newBuyer.id;
    }
}
```

Recommendation:

We advise the code to mimic the statements of `BuyerBase::createBuyerInternal`, simply validating that the `_buyer` is a non-zero address. The optimization can also be achieved by relocating the "shared" statements between the two implementations to a common `internal` function, easing code maintenance while achieving the optimization described.

Alleviation (9ed49d780f04b2f168001f72d6c09f5fc94818b7):

The security checks have been removed from the `BuyerBase::createBuyerInternal` function and were instead relocated to the `BuyerHandlerFacet::createBuyer` function.

As a result, the code of the Boson Protocol has remained functionally identical whilst its inefficiencies have been eliminated rendering this exhibit fully addressed.

FundsLib Code Style Findings

FLB-01C: Redundant Self-Assignment

Type	Severity	Location
Gas Optimization	Informational	FundsLib.sol:L345

Description:

The referenced statement represents an assignment-to-self of the `sellerRoyalties` variable.

Example:

contracts/protocol/libs/FundsLib.sol

```sol
sellerRoyalties = sellerRoyalties;
```

Recommendation:

We advise the statement to be properly omitted, optimizing the code's gas cost.

Alleviation (9ed49d780f04b2f168001f72d6c09f5fc94818b7):

The redundant self-assignment has been omitted, optimizing the code's gas cost.

FLB-02C: Repetitive Value Literal

Type	Severity	Location
Code Style	Informational	FundsLib.sol:L260, L275, L281, L321, L323, L344, L565, L568, L584

Description:

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

Example:

contracts/protocol/libs/FundsLib.sol

SOL

```
260 effectivePriceMultiplier = 10000;
```

Recommendation:

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

Alleviation (9ed49d780f04b2f168001f72d6c09f5fc94818b7):

The repetitive value literal has been relocated to a `BosonConstants` level `constant` labelled `HUNDRED_PERCENT`, increasing the legibility of the codebase.

OfferHandlerFacet Code Style Findings

OHF-01C: Redundant Application of Security Modifier

Type	Severity	Location
Gas Optimization	Informational	OfferHandlerFacet.sol:L336

Description:

The `PausableBase::offersNotPaused` security modifier will be redundantly applied by both the top-level **`OfferHandlerFacet::updateOfferRoyaltyRecipientsBatch`** function as well as each invocation of the **`OfferHandlerFacet::updateOfferRoyaltyRecipients`** it performs.

Example:

contracts/protocol/facets/OfferHandlerFacet.sol

SOL

```
/**
 * @notice Sets new valid royalty info.
 *
 * Emits an OfferRoyaltyInfoUpdated event if successful.
 *
 * Reverts if:
 * - The offers region of protocol is paused
 * - Offer does not exist
 * - Caller is not the assistant of the offer
 * - New royalty info is invalid
```

Example (Cont.):

```sol
 *
 *  @param _offerId - the id of the offer to be updated
 *  @param _royaltyInfo - new royalty info
 */
function updateOfferRoyaltyRecipients(
    uint256 _offerId,
    RoyaltyInfo calldata _royaltyInfo
) public override offersNotPaused nonReentrant {
    // Make sure the caller is the assistant, offer exists and is not voided
    Offer storage offer = getValidOfferWithSellerCheck(_offerId);

    validateRoyaltyInfo(protocolLookups(), protocolLimits(), offer.sellerId, _royaltyInfo);

    // Add new entry to the royaltyInfo array
    offer.royaltyInfo.push(_royaltyInfo);

    // Notify watchers of state change
    emit OfferRoyaltyInfoUpdated(_offerId, offer.sellerId, _royaltyInfo, msgSender());
}

/**
 * @notice Sets new valid until date for a batch of offers.
 *
 * Emits an OfferExtended event for every offer if successful.
 *
 * Reverts if:
 * - The offers region of protocol is paused
 * - For any of the offers:
```

Example (Cont.):

```sol
 *   - Offer does not exist
 *   - Caller is not the assistant of the offer
 *   - New royalty info is invalid
 *
 *  @param _offerIds - list of ids of the offers to extend
 *  @param _royaltyInfo - new royalty info
 */
function updateOfferRoyaltyRecipientsBatch(
    uint256[] calldata _offerIds,
    BosonTypes.RoyaltyInfo calldata _royaltyInfo
) external override offersNotPaused {
    for (uint256 i = 0; i < _offerIds.length; ) {
        updateOfferRoyaltyRecipients(_offerIds[i], _royaltyInfo);

        unchecked {
            i++;
        }
    }
}
```

Recommendation:

We advise the code of `OfferHandlerFacet::updateOfferRoyaltyRecipients` to be relocated to an `internal` function that is invoked by both the function itself as well as the `OfferHandlerFacet::updateOfferRoyaltyRecipientsBatch` function, applying the `ReentrancyGuardBase::nonReentrant` modifier while optimizing the `PausableBase::offersNotPaused` modifier's application.

Alleviation (9ed49d780f04b2f168001f72d6c09f5fc94818b7):

Our recommendation was applied to the referenced function as well as to several others, suffixing their `internal` implementation with `Internal` and applying the relevant security checks in top-level `external` functions.

As such, we consider this optimization fully applied.

PriceDiscoveryBase Code Style Findings

PDB-01C: Inexistent Error Message

Type	Severity	Location
Code Style	Informational	PriceDiscoveryBase.sol:L279

Description:

The linked `require` check has no error message explicitly defined.

Example:

contracts/protocol/bases/PriceDiscoveryBase.sol

SOL

```
279 require(protocolNativeBalanceAfter == protocolNativeBalanceBefore);
```

Recommendation:

We advise one to be set so to increase the legibility of the codebase and aid in validating the `require` check's condition.

Alleviation (9ed49d780f04b2f168001f72d6c09f5fc94818b7):

The referenced `require` check is no longer present in the codebase, rendering this exhibit no longer applicable.

SellerHandlerFacet Code Style Findings

SHF-01C: Inefficient `mapping` Lookups

Type	Severity	Location
Gas Optimization	Informational	SellerHandlerFacet.sol:L493, L500, L562, L568, L571, L631, L637

Description:

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

Example:

contracts/protocol/facets/SellerHandlerFacet.sol

SOL

```sol
562 uint256 royaltyRecipientIndex = lookups.royaltyRecipientIndexBySellerAndRecipient[_sellerId][
563     _royaltyRecipients[i].wallet
564 ];
565
566 if (royaltyRecipientIndex == 0) {
567     // update index
568     lookups.royaltyRecipientIndexBySellerAndRecipient[_sellerId][_royaltyRecipients[i].wallet] =
569         royaltyRecipientId +
570         1;
571     delete lookups.royaltyRecipientIndexBySellerAndRecipient[_sellerId][
```

Example (Cont.):

SOL

```
        royaltyRecipients[royaltyRecipientId].wallet
    ];
} else {
```

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.

Alleviation (9ed49d780f04b2f168001f72d6c09f5fc94818b7):

All referenced inefficient `mapping` lookups have been optimized to the greatest extent possible, significantly reducing the gas cost of the functions the statements were located in.

SHF-02C: Non-Uniform Royalty Recipient ID Definition

Type	Severity	Location
Standard Conformity	Informational	SellerHandlerFacet.sol:L555

Description:

The royalty recipient IDs are 1-based in the Boson Protocol system, however, they are expected to be passed in as 0-based in the `SellerHandlerFacet::updateRoyaltyRecipients` function.

Example:

contracts/protocol/facets/SellerHandlerFacet.sol

SOL

```
function updateRoyaltyRecipients(
    uint256 _sellerId,
    uint256[] calldata _royaltyRecipientIds,
    RoyaltyRecipient[] calldata _royaltyRecipients
) external sellersNotPaused nonReentrant {
    // Cache protocol lookups and sender for reference
    ProtocolLib.ProtocolLookups storage lookups = protocolLookups();

    // Make sure admin is the caller and get the seller
    address treasury;
```

Example (Cont.):

SOL

```solidity
    {
        (Seller storage seller, ) = validateAdminStatus(lookups, _sellerId);
        treasury = seller.treasury;
    }

    if (_royaltyRecipientIds.length != _royaltyRecipients.length) revert ArrayLengthMismatch();

    RoyaltyRecipient[] storage royaltyRecipients = lookups.royaltyRecipientsBySeller[_sellerId];
    // uint256 royaltyRecipientIdsLength = _royaltyRecipientIds.length; // TODO can be optimized?
    uint256 royaltyRecipientsLength = royaltyRecipients.length;
    for (uint256 i = 0; i < _royaltyRecipientIds.length; ) {
        uint256 royaltyRecipientId = _royaltyRecipientIds[i];

        if (royaltyRecipientId >= royaltyRecipientsLength) revert InvalidRoyaltyRecipientId();
```

Recommendation:

Given that the current implementation is efficient, we advise proper documentation to be introduced to the `SellerHandlerFacet::updateRoyaltyRecipients` function indicating that the IDs are meant to be passed in as 0-based.

Alleviation (9ed49d780f04b2f168001f72d6c09f5fc94818b7):

The documentation was updated to reflect that the functions expect `0` based indexes as input arguments, addressing this exhibit.

SequentialCommitHandlerFacet Code Style Findings

SCH-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	SequentialCommitHandlerFacet.sol:L144

Description:

The linked mathematical operation is guaranteed to be performed safely by surrounding conditionals evaluated in either `require` checks or `if-else` constructs.

Example:

contracts/protocol/facets/SequentialCommitHandlerFacet.sol

SOL

```
uint256 currentPrice = len == 0 ? offer.price : exchangeCosts[len - 1].price;
```

Recommendation:

Given that safe arithmetics are toggled on by default in `pragma` versions of `0.8.X`, we advise the linked statement to be wrapped in an `unchecked` code block thereby optimizing its execution cost.

Alleviation (9ed49d780f04b2f168001f72d6c09f5fc94818b7):

An `unchecked` code block has been properly introduced to the referenced segment, optimizing its gas cost.

Finding Types

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

Input Sanitization

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

Indeterminate Code

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

Language Specific

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

Curve Specific

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

Code Style

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

Mathematical Operations

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

Logical Fault

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

Privacy Concern

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

Proof Concern

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

Severity Definition

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

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

Severity Levels

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

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

Unknown Severity

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

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

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

Informational Severity

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

Minor Severity

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

Medium Severity

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

Major Severity

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

Likelihood & Impact Assessment

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

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

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

Disclaimer

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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