OMNISCIA

November 1, 2022

# SMART CONTRACT AUDIT REPORT

Boson Protocol
v2.1



omniscia.io


info@omniscia.io



Online report: boson-protocol-version-2

# Version 2 Security Audit

## Audit Overview

*Disclaimer: The highlighted issues of the following summary reflected an understanding of the codebase at the time the audit was performed and were inadequately conveyed to the Boson Protocol team by the Omniscia team. For more information, consult the **Amendment of Report Summary** chapter that follows.*

We were tasked with performing another audit of the Boson Protocol and in particular their V2 implementation that revised the initial codebase to now utilize the EIP-2535 Diamond standard as well as compartmentalized code into several different entities that together form the Boson Voucher system.

Over the course of the audit, we identified multiple issues of varying severity revolving around edge cases that the system is inadequately equipped to deal with as well as certain medium issues with regards to the cryptographic implementation used by the system.

The code has been developed to a high standard, however, we have observed certain patterns that we advise against such as input validation being split between both `internal` and `public` facing functions (i.e. `raiseDisputeInternal` of `DisputeBase` expects a state different than `Redeemed` but is validated at the invocation level) and the presence of natural race conditions that should otherwise be avoided.

While the former type of issue does not currently pose an active security threat, it can easily evolve into one based on the programming practices as well as the expected deviation between each contributor's coding style.

With regards to the latter, we specifically refer to the creation of the various account types (buyer & seller) which allow arbitrary addresses to be specified. An unsuspecting user's "seller" account may be created with values different than the ones they specified in the creation call due to a malicious user's transaction containing the same input parameters with slight deviations and a higher transaction fee.

We strongly encourage the Boson Protocol team to reconsider the permissionless-ness of the account creation functions as they may lead to fraud attempts based on different seller configurations due to the difference between what was submitted in the UI vs what was ultimately executed on-chain due to the aforementioned race condition.

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 evaluated all exhibits identified within the report and provided an alleviation for each one in the form of a detailed PDF document as well as rationale as to why certain exhibits will remain unaddressed.

All exhibits have either been adequately dealt with, properly nullified, or responsibly acknowledged by the Boson Protocol team and no outstanding issues identified in the report remain in the codebase.

## Amendment of Report Summary

After extensive discussions with the Boson Protocol team in relation to our original audit summary, we concluded that certain aspects of the codebase required a re-visit by the Boson Protocol team as they were inadequately conveyed by us.

In detail, the split of validation logic within the code was a concious and deliberate choice by the Boson Protocol team as the same checks are not required wherever the `internal`-style functions are invoked thus allowing the `public` facing functions to apply any additional input sanitization on an as-needed basis, however, the comments of the `internal` functions were misleading. This was alleviated in finding DBE-01M.

With regards to the permission-less account creation, we incorrectly assumed that this was a design choice that the Boson Protocol team was aware of and thus simply commented on it rather than formulating it as a finding.

Once the Boson Protocol team evaluated the audit's summary in detail, they detected that the issues outlined within it were valid and as such should be integrated in the report as they proceeded to alleviate them.

We have thus added the incorrect comments as well as the race-condition issues as separate exhibits in the report and the Boson Protocol team performed extensive updates to correct both thus rendering them no longer applicable to the codebase.

On a final note, the Boson Protocol team assessed that the race-condition issue described for the seller and buyer accounts pose no threat to buyer creation, however, the additional case of a dispute resolver creation required this form of race-condition protection as well.

They have proceeded to amend that portion of the codebase similarly to the seller creation process with the same opt-in system described in SBE-01M.

## Contracts Assessed

| Files in Scope | Repository | Commit(s) |
| --- | --- | --- |
| AccessController.sol (ACR) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| AccountHandlerFacet.sol (AHF) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| BuyerBase.sol (BBE) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| BosonTypes.sol (BTS) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| BundleBase.sol (BBS) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| BosonVoucher.sol (BVR) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| BosonConstants.sol (BCS) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |

| Files in Scope | Repository | Commit(s) |
| --- | --- | --- |
| BeaconClientLib.sol (BCL) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| BeaconClientBase.sol (BCB) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| BeaconClientProxy.sol (BCP) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| BundleHandlerFacet.sol (BHF) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| ClientLib.sol (CLB) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| ClientBase.sol (CBE) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| ClientProxy.sol (CPY) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |

| Files in Scope | Repository | Commit(s) |
| --- | --- | --- |
| ConfigHandlerFacet.sol (CHF) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| ClientExternalAddressesBase.sol (CEA) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| DiamondLib.sol (DLB) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| DisputeBase.sol (DBE) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| DiamondCutFacet.sol (DCF) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| DiamondLoupeFacet.sol (DLF) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| DisputeHandlerFacet.sol (DHF) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |

| Files in Scope | Repository | Commit(s) |
| --- | --- | --- |
| EIP712Lib.sol (EIP) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| ERC165Facet.sol (ERC) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| ExchangeHandlerFacet.sol (EHF) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| FundsLib.sol (FLB) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| FundsHandlerFacet.sol (FHF) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| GroupBase.sol (GBE) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| GroupHandlerFacet.sol (GHF) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |

| Files in Scope | Repository | Commit(s) |
| --- | --- | --- |
| JewelerLib.sol (JLB) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| MetaTransactionsHandlerFacet.sol (MTH) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| OfferBase.sol (OBE) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| OfferHandlerFacet.sol (OHF) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| OrchestrationHandlerFacet.sol (CON) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| Proxy.sol (PRO) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| ProtocolLib.sol (PLB) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |

| Files in Scope | Repository | Commit(s) |
| --- | --- | --- |
| PausableBase.sol (PBE) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| ProtocolBase.sol (PBS) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| ProtocolDiamond.sol (PDD) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| ReentrancyGuardBase.sol (RGB) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| SellerBase.sol (SBE) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| SellerHandlerFacet.sol (SHF) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |
| TwinBase.sol (TBE) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |

| Files in Scope | Repository | Commit(s) |
| --- | --- | --- |
| TwinHandlerFacet.sol (THF) | boson-protocol-contracts | 25ea648255, 44009967e4, 6dae5d2602, 68ebb15f14, f62f5f26c2 |

## Audit Synopsis

| Severity | Identified | Alleviated | Partially Alleviated | Acknowledged |
| --- | --- | --- | --- | --- |
| Unknown | 8 | 8 | 0 | 0 |
| Informational | 43 | 37 | 0 | 6 |
| Minor | 14 | 10 | 0 | 4 |
| Medium | 5 | 5 | 0 | 0 |
| Major | 0 | 0 | 0 | 0 |

During the audit, we filtered and validated a total of **13 findings utilizing static analysis** tools as well as identified a total of **57 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 introduce potential misbehaviours of the system as well as exploits.

The list below covers each segment of the audit in depth and links to the respective chapter of the report:

- 💻 **Compilation**
- 🔍 **Static Analysis**
- 👁 **Manual Review**
- 🖌 **Code Style**

# Compilation

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

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

```
BASH

npx hardhat compile
```

The `hardhat` tool automatically selects Solidity version `0.8.9` based on the version specified within the `hardhat.config.js` file.

The project contains discrepancies with regards to the Solidity version used as the `pragma` statements of the contracts are open-ended (`^0.8.0`).

We advise them to be locked to `0.8.9` (`=0.8.9`), the same version utilized for our static analysis as well as optimizational review of the codebase.

During compilation with the `hardhat` pipeline, no errors were identified that relate to the syntax or bytecode size of the contracts.

# Static Analysis

The execution of our static analysis toolkit identified **747 potential issues** within the codebase of which **716 were ruled out to be false positives** or negligible findings.

The remaining **31 issues** were validated and grouped and formalized into the **13 exhibits** that follow:

| ID | Severity | Addressed | Title |
|---|---|---|---|
| **BVR-01S** | Informational | Acknowledged | Illegible Numeric Value Representation |
| **CEA-01S** | Minor | Yes | Incorrect `payable` Specifier |
| **CEA-02S** | Minor | Acknowledged | Inexistent Sanitization of Input Addresses |
| **CPY-01S** | Minor | Yes | Incorrect `payable` Specifier |
| **CHF-01S** | Informational | Acknowledged | Illegible Numeric Value Representations |
| **CHF-02S** | Minor | Yes | Inexistent Sanitization of Input Addresses |
| **DLF-01S** | Informational | Yes | Redundant Variable Assignments |
| **DHF-01S** | Informational | Acknowledged | Illegible Numeric Value Representations |
| **FLB-01S** | Informational | Acknowledged | Illegible Numeric Value Representation |
| **FLB-02S** | Medium | Yes | Improper Invocations of EIP-20 `transfer` / `transferFrom` |
| **OBE-01S** | Informational | Acknowledged | Illegible Numeric Value Representations |
| **PDD-01S** | Minor | Yes | Inexistent Sanitization of Input Address |
| **PLB-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 specialized voucher system based on the Diamond standard.

As the project at hand implements a diamond-standard based specialized voucher system, intricate care was put into ensuring that the **flow of assets & funds within the system conforms to the specifications and restrictions** laid forth within the protocol's specification, that the codebase **resembles the original V1 implementation in execution flows**, and that the various **components of the Diamond standard properly interact between them**.

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 misbehaviours and potential vulnerabilities** within the system which could have had **moderate-to-severe ramifications** to its overall operation under **edge-case circumstances**, however, they were conveyed ahead of time to the Boson Protocol team to be **promptly remediated**.

Additionally, the system was investigated for any other commonly present attack vectors such as re-entrancy attacks, mathematical truncations, logical flaws and **ERC / EIP** standard inconsistencies. The documentation of the project was satisfactory to a great extent, however, the terminology used around certain modules such as the "twin", "bundle", and "group" terms can become slightly ambiguous and we advise a "glossary" of terms to be provided within the repository to better aid the code's legibility.

A total of **57 findings** were identified over the course of the manual review of which **21 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 |
|---|---|---|---|
| ACR-01M | Unknown | Yes | Potentially Incorrect Dependency Usage |
| ACR-02M | Minor | Acknowledged | Circular Access Control Dependency |
| BCS-01M | Unknown | Nullified | Potentially Incorrect Type-Hash Definitions |
| BVR-01M | Unknown | Yes | Inexistent Prevention of Self-Transfer |
| BBS-01M | Minor | Yes | Inexistent Validation of Valid Bundle Creation |
| CLB-01M | Unknown | Yes | Potentially Discrepant Access Control |
| CHF-01M | Unknown | Yes | Potentially Overly Centralized Configuration Control |
| CHF-02M | Minor | Yes | Inexistent Validation of Numeric Configuration |
| DBE-01M | Unknown | Yes | Potentially Misleading Validation Comments |
| EIP-01M | Unknown | Yes | Inexistent Validation of Meta-TX Sender Validity |
| EIP-02M | Minor | Yes | Improper Domain Separator Retrieval |
| EIP-03M | Medium | Yes | Insecure Elliptic Curve Recovery Mechanism |
| EHF-01M | Medium | Nullified | Improper Invocation of EIP-20 `transfer` |
| FHF-01M | Minor | Acknowledged | Inexistent Validation of Token Type |
| GBE-01M | Minor | Nullified | Inexistent Validation of Group Entries |
| GBE-02M | Minor | Yes | Insufficient Evaluation of `maxOffersPerGroup` |

| ID | Severity | Addressed | Title |
| --- | --- | --- | --- |
| MTH-01M | Medium | Yes | Account Agnostic Nonce System |
| OBE-01M | Minor | Acknowledged | Inexistent Validation of Proper Voucher Redemption Setup |
| OHF-01M | Minor | Yes | Inexistent Re-Entrancy Protection |
| PBS-01M | Unknown | Yes | Potentially Unsafe Existence Check |
| SBE-01M | Medium | Yes | Potential of Race Condition in Seller Creation |

# Code Style

During the manual portion of the audit, we identified **36 optimizations** that can be applied to the codebase that will decrease the gas-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 |
|---|---|---|---|
| BCS-01C | Informational | Yes | Generic Typographic Mistakes |
| BTS-01C | Informational | Nullified | Potentially Harmful Default Enum State |
| BVR-01C | Informational | Yes | Hard-Coded Interface Type |
| BBS-01C | Informational | Yes | Repetitive Invocations of Getter Function |
| BHF-01C | Informational | Yes | Unused Code |
| DLB-01C | Informational | Yes | Redundant Conditional Return Statement |
| DBE-01C | Informational | Yes | Repetitive Invocations of Getter Function |
| EIP-01C | Informational | Yes | Deprecated Chain ID Retrieval Code |
| EHF-01C | Informational | Yes | Multiple Top-Level Type Declarations |
| FHF-01C | Informational | Yes | Inefficient `mapping` Lookups |
| FHF-02C | Informational | Yes | Repetitive Invocations of Getter Functions |
| FHF-03C | Informational | Yes | Suboptimal Struct Declaration Style |
| FLB-01C | Informational | Yes | Ineffectual Usage of Safe Arithmetics |
| FLB-02C | Informational | Yes | Inefficient `mapping` Lookups |
| FLB-03C | Informational | Yes | Inexplicable Contract Specifier |
| GBE-01C | Informational | Yes | Repetitive Invocations of Getter Functions |

| ID | Severity | Addressed | Title |
|---|---|---|---|
| GHF-01C | Informational | Yes | Inefficient `mapping` Lookups |
| GHF-02C | Informational | Yes | Repetitive Invocations of Getter Function |
| OBE-01C | Informational | Yes | Suboptimal Struct Declaration Style |
| CON-01C | Informational | Yes | Redundant Modifier Overlap |
| PBE-01C | Informational | Yes | Bytecode Size Optimization |
| PBE-02C | Informational | Yes | Potential Operator Optimization |
| PBS-01C | Informational | Yes | Repetitive Invocations of Getter Functions |
| PLB-01C | Informational | Yes | Generic Typographic Mistake |
| PLB-02C | Informational | Yes | Inefficient Data Types |
| RGB-01C | Informational | Yes | Outdated Documentation of Reentrancy Guard |
| SBE-01C | Informational | Yes | Multiple Top-Level Type Declarations |
| SBE-02C | Informational | Yes | Repetitive Invocations of Getter Functions |
| SHF-01C | Informational | Yes | Inefficient `mapping` Lookups |
| SHF-02C | Informational | Yes | Repetitive Invocations of Getter Function |
| TBE-01C | Informational | Yes | Hard-Coded Interface Types |
| TBE-02C | Informational | Yes | Inefficient Loop Limit Evaluations |

| ID | Severity | Addressed | Title |
| --- | --- | --- | --- |
| TBE-03C | Informational | Yes | Inefficient `mapping` Lookups |
| TBE-04C | Informational | Yes | Repetitive Invocations of Getter Function |
| TBE-05C | Informational | Yes | Suboptimal Struct Declaration Type |
| THF-01C | Informational | Yes | Repetitive Invocations of Getter Function |

# BosonVoucher Static Analysis Findings

## BVR-01S: Illegible Numeric Value Representation

| Type | Severity | Location |
| --- | --- | --- |
| **Code Style** | Informational | BosonVoucher.sol:L212 |

### Description:

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

### Example:

contracts/protocol/clients/voucher/BosonVoucher.sol

SOL

```
212 royaltyAmount = (_salePrice * _royaltyPercentage) / 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 (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The Boson Protocol team evaluated this exhibit and opted to retain the current representational style in the codebase as they deem an underscore in small numbers such as the value `100_00` as unnecessary. As a result, we consider this exhibit as acknowledged.

# ClientExternalAddressesBase Static Analysis Findings

## CEA-01S: Incorrect `payable` Specifier

| Type | Severity | Location |
| --- | --- | --- |
| Language Specific | Minor | ClientExternalAddressesBase.sol:L31 |

### Description:

The `constructor` of the `ClientExternalAddressesBase` contract is set as `payable` incorrectly.

### Impact:

Funds can currently be permanently locked in the contract in case they are sent during its construction.

### Example:

contracts/protocol/bases/ClientExternalAddressesBase.sol

SOL

```sol
27  constructor(
28      address _accessController,
29      address _protocolAddress,
30      address _impl
31  ) payable {
32      // Get the ProxyStorage struct
33      ClientLib.ProxyStorage storage ps = ClientLib.proxyStorage();
34
35      // Store the AccessController address
36      ps.accessController = IAccessControlUpgradeable(_accessController);
37
38      // Store the Protocol Diamond address
39      ps.protocolDiamond = _protocolAddress;
40
41      // Store the implementation address
42      ps.implementation = _impl;
43  }
```

### Recommendation:

We advise the `payable` modifier to be omitted avoiding lock of funds.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The `payable` specifier from the `constructor` has been properly omitted from the codebase as advised.

# CEA-02S: Inexistent Sanitization of Input Addresses

| Type | Severity | Location |
| --- | --- | --- |
| **Input Sanitization** | Minor | ClientExternalAddressesBase.sol:L27-L43, L60-L69, L85-L94, L116-L125 |

### Description:

The linked function(s) accept `address` arguments yet do not properly sanitize them.

### 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/ClientExternalAddressesBase.sol

SOL

```sol
constructor(
    address _accessController,
    address _protocolAddress,
    address _impl
) payable {
    // Get the ProxyStorage struct
    ClientLib.ProxyStorage storage ps = ClientLib.proxyStorage();

    // Store the AccessController address
    ps.accessController = IAccessControlUpgradeable(_accessController);

    // Store the Protocol Diamond address
    ps.protocolDiamond = _protocolAddress;

    // Store the implementation address
    ps.implementation = _impl;
}
```

### Recommendation:

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

### Alleviation (44009967e4):

The now-two input arguments are correctly sanitized in the first referenced instance (the `constructor`), however, the remaining referenced instances remain unaddressed rendering this exhibit partially dealt with.

### Alleviation (6dae5d2602):

Input sanitization has now been introduced to the `setImplementation` and `setProtocolAddress` functions whilst the `setAccessController` function still allows a zero-value input address which is intended functionality based on the Boson Protocol team's response. As a result, we consider this exhibit fully alleviated.

# ClientProxy Static Analysis Findings

## CPY-01S: Incorrect `payable` Specifier

| Type | Severity | Location |
| --- | --- | --- |
| **Language Specific** | Minor | ClientProxy.sol:L26-L30 |

### Description:

The `constructor` of the `ClientProxy` contract is set as `payable` incorrectly.

### Impact:

Funds can currently be permanently locked in the contract in case they are sent during its construction.

### Example:

contracts/protocol/clients/proxy/ClientProxy.sol

SOL

```
26  constructor(
27      address _accessController,
28      address _protocolAddress,
29      address _impl
30  ) payable ClientExternalAddressesBase(_accessController, _protocolAddress, _impl) {}
```

### Recommendation:

We advise the `payable` modifier to be omitted avoiding lock of funds.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The `payable` specifier from the `constructor` has been properly omitted from the codebase as advised.

# ConfigHandlerFacet Static Analysis Findings

## CHF-01S: Illegible Numeric Value Representations

| Type | Severity | Location |
| --- | --- | --- |
| **Code Style** | Informational | ConfigHandlerFacet.sol:L159, L379, L409, L463 |

### Description:

The linked representations of numeric literals are sub-optimally represented decreasing the legibility of the codebase.

### Example:

contracts/protocol/facets/ConfigHandlerFacet.sol

SOL

```
159 require(_protocolFeePercentage <= 10000, FEE_PERCENTAGE_INVALID);
```

### Recommendation:

To properly illustrate each 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 (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The Boson Protocol team evaluated this exhibit and opted to retain the current representational style in the codebase as they deem an underscore in small numbers such as the value `100_00` as unnecessary. As a result, we consider this exhibit as acknowledged.

## CHF-02S: Inexistent Sanitization of Input Addresses

| Type | Severity | Location |
| --- | --- | --- |
| **Input Sanitization** | Minor | ConfigHandlerFacet.sol:L76-L79, L95-L98, L114-L117, L133-L136, L490-L500 |

### Description:

The linked function(s) accept `address` arguments yet do not properly sanitize them.

### 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/facets/ConfigHandlerFacet.sol

SOL

```sol
114 function setVoucherBeaconAddress(address _voucherBeaconAddress) public override onlyRo
115     protocolAddresses().voucherBeacon = _voucherBeaconAddress;
116     emit VoucherBeaconAddressChanged(_voucherBeaconAddress, msgSender());
117 }
```

### Recommendation:

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

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

All referenced instances of `address` inputs are now properly sanitized against the zero address as advised.

# DiamondLoupeFacet Static Analysis Findings

## DLF-01S: Redundant Variable Assignments

| Type | Severity | Location |
| --- | --- | --- |
| **Gas Optimization** | Informational | DiamondLoupeFacet.sol:L40, L52, L132, L140 |

### Description:

The linked variables are assigned to redundantly to the default value of each relevant data type (i.e. `uint256` assigned to `0`, `address` assigned to `address(0)` etc.).

### Example:

contracts/diamond/facets/DiamondLoupeFacet.sol

SOL

```
40  bool continueLoop = false;
```

### Recommendation:

We advise the assignments to be safely omitted optimizing the codebase.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

All four referenced redundant assignments have been safely omitted from the codebase.

# DisputeHandlerFacet Static Analysis Findings

## DHF-01S: Illegible Numeric Value Representations

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | DisputeHandlerFacet.sol:L243, L386 |

### Description:

The linked representations of numeric literals are sub-optimally represented decreasing the legibility of the codebase.

### Example:

contracts/protocol/facets/DisputeHandlerFacet.sol

SOL

```
243 require(_buyerPercent <= 10000, INVALID_BUYER_PERCENT);
```

### Recommendation:

To properly illustrate each 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 (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The Boson Protocol team evaluated this exhibit and opted to retain the current representational style in the codebase as they deem an underscore in small numbers such as the value `100_00` as unnecessary. As a result, we consider this exhibit as acknowledged.

# FundsLib Static Analysis Findings

## FLB-01S: Illegible Numeric Value Representation

| Type | Severity | Location |
| --- | --- | --- |
| **Code Style** | Informational | FundsLib.sol:L181 |

### 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

```sol
181 buyerPayoff = (pot * dispute.buyerPercent) / 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 (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The Boson Protocol team evaluated this exhibit and opted to retain the current representational style in the codebase as they deem an underscore in small numbers such as the value `100_00` as unnecessary. As a result, we consider this exhibit as acknowledged.

## FLB-02S: Improper Invocations of EIP-20 `transfer` / `transferFrom`

| Type | Severity | Location |
| --- | --- | --- |
| **Standard Conformity** | Medium | FundsLib.sol:L228, L269 |

### Description:

The linked statements do not properly validate the returned `bool` values of the **EIP-20** standard `transfer` & `transferFrom` functions. As the **standard dictates**, callers **must not** assume that `false` is never returned.

### Impact:

If the token utilized can return a `false` value under certain conditions but the code does not validate it, the contract itself can be compromised as having received / sent funds that it never did.

### Example:

contracts/protocol/libs/FundsLib.sol

SOL

```sol
// transfer ERC20 tokens from the caller
try IERC20(_tokenAddress).transferFrom(EIP712Lib.msgSender(), address(this), _amount)
    bytes memory error
) {
    string memory reason = error.length == 0 ? TOKEN_TRANSFER_FAILED : string(error);
    revert(reason);
}
```

### Recommendation:

Since not all standardized tokens are **EIP-20** compliant (such as Tether / USDT), we advise a safe wrapper library to be utilized instead such as `SafeERC20` by OpenZeppelin to opportunistically validate the returned `bool` only if it exists in each instance. The `safeTransfer` / `safeTransferFrom` invocations can consequently be wrapped in the existing `try-catch` blocks to properly illustrate custom error reasons in such cases.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

Both `transfer` and `transferFrom` invocations have been properly replaced by their `safe`-prefixed counterparts from the OpenZeppelin library alleviating this exhibit in full.

# OfferBase Static Analysis Findings

## OBE-01S: Illegible Numeric Value Representations

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | OfferBase.sol:L177, L201, L204, L206 |

### Description:

The linked representations of numeric literals are sub-optimally represented decreasing the legibility of the codebase.

### Example:

contracts/protocol/bases/OfferBase.sol

SOL

```
177 (feeAmount * pl.buyerEscalationDepositPercentage) / 10000
```

### Recommendation:

To properly illustrate each 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 (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The Boson Protocol team evaluated this exhibit and opted to retain the current representational style in the codebase as they deem an underscore in small numbers such as the value `100_00` as unnecessary. As a result, we consider this exhibit as acknowledged.

# ProtocolDiamond Static Analysis Findings

## PDD-01S: Inexistent Sanitization of Input Address

| Type | Severity | Location |
| --- | --- | --- |
| **Input Sanitization** | Minor | ProtocolDiamond.sol:L35-L39 |

### 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/diamond/ProtocolDiamond.sol

SOL

```sol
35  constructor(
36      IAccessControlUpgradeable _accessController,
37      IDiamondCut.FacetCut[] memory _facetCuts,
38      bytes4[] memory _interfaceIds
39  ) payable {
40      // Get the DiamondStorage struct
41      DiamondLib.DiamondStorage storage ds = DiamondLib.diamondStorage();
42
43      // Set the AccessController instance
44      ds.accessController = _accessController;
45
46      // Cut the diamond with the given facets
47      JewelerLib.diamondCut(_facetCuts, address(0), new bytes(0));
48
49      // Add supported interfaces
50      if (_interfaceIds.length > 0) {
51          for (uint8 x = 0; x < _interfaceIds.length; x++) {
52              DiamondLib.addSupportedInterface( interfaceIds[x]);
```

```
53            }
54        }
55    }
```

### Recommendation:

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

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The `_accessController` input address is now properly sanitized in the referenced statements.

# ProtocolLib Static Analysis Findings

## PLB-01S: Illegible Numeric Value Representation

| Type | Severity | Location |
| --- | --- | --- |
| **Code Style** | Informational | ProtocolLib.sol:L66 |

### Description:

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

### Example:

contracts/protocol/libs/ProtocolLib.sol

SOL

```
66  uint16 percentage; // 1.75% = 175, 100% = 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 (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The Boson Protocol team evaluated this exhibit and opted to retain the current representational style in the codebase as they deem an underscore in small numbers such as the value `100_00` as unnecessary. As a result, we consider this exhibit as acknowledged.

# AccessController Manual Review Findings

## ACR-01M: Potentially Incorrect Dependency Usage

| Type | Severity | Location |
| --- | --- | --- |
| **Language Specific** | Unknown | AccessController.sol:L12 |

### Description:

The contract inherits from the `AccessControlUpgradeable` contract yet does not invoke its `__AccessControl_init` initialization hook nor does the contract itself represent an upgrade-able implementation as it contains logic in its `constructor`.

### Example:

contracts/access/AccessController.sol

SOL

```sol
// SPDX-License-Identifier: GPL-3.0-or-later
pragma solidity ^0.8.0;

import "../domain/BosonConstants.sol";
import "@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol";

/**
 * @title AccessController
 *
 * @notice Implements centralized role-based access for Boson Protocol contracts.
 */
contract AccessController is AccessControlUpgradeable {
    /**
     * @notice Constructor
     *
     * Grants ADMIN role to deployer.
     * Sets ADMIN as role admin for all other roles.
     */
    constructor() {
        _setupRole(ADMIN, msg.sender);
        _setRoleAdmin(ADMIN, ADMIN);
        _setRoleAdmin(PAUSER, ADMIN);
```

```
23          _setRoleAdmin(PROTOCOL, ADMIN);
24          _setRoleAdmin(CLIENT, ADMIN);
25          _setRoleAdmin(UPGRADER, ADMIN);
26          _setRoleAdmin(FEE_COLLECTOR, ADMIN);
27      }
28  }
```

### Recommendation:

We advise the dependency of the contract to be assessed and either adjusted to the non-upgrade-able one or the contract itself to be adjusted to conform to the upgrade-able contract standards and not contain any logic in its `constructor`.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The non-upgradeable and now-local `AccessControl` dependency is inherited by the contract instead of the original upgradeable one thus alleviating this exhibit as its concerns are no longer relevant.

# ACR-02M: Circular Access Control Dependency

| Type | Severity | Location |
| --- | --- | --- |
| **Standard Conformity** | Minor | AccessController.sol:L21 |

### Description:

The `ADMIN` role contains a circular administration meaning that a single `ADMIN` role holder can completely compromise the system and remove all other members.

### Impact:

A single rogue `ADMIN` role holder can compromise the system entirely as they are capable of removing other `ADMIN` holders as well as assigning all roles defined in the `AccessController` contract.

### Example:

contracts/access/AccessController.sol

SOL

```
21  _setRoleAdmin(ADMIN, ADMIN);
```

### Recommendation:

We advise this trait of the system to be re-evaluated and the `ADMIN` role to potentially be held by a single entity throughout the `AccessController` contract's lifetime.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The Boson Protocol team has opted to retain the current hierarchical structure in place as they wish to retain the ability to transfer `ADMIN` ownership to multiple parties, initially the multi-signature wallet and later a dedicated DAO module that is planned for the future. Given that alternative albeit a bit more complex workflows do exist to at least ensure a singular `ADMIN` member remains active at any given moment, we consider this exhibit as acknowledged.

# BosonConstants Manual Review Findings

## BCS-01M: Potentially Incorrect Type-Hash Definitions

| Type | Severity | Location |
| --- | --- | --- |
| **Standard Conformity** | Unknown | BosonConstants.sol:L177, L181, L187, L193 |

### Description:

The referenced type-hash definitions appear to contain two different definitions in sequence thereby not conforming to the type-hash paradigm.

### Example:

contracts/domain/BosonConstants.sol

SOL

```
176 bytes32 constant META_TX_COMMIT_TO_OFFER_TYPEHASH = keccak256(
177     "MetaTxCommitToOffer(uint256 nonce,address from,address contractAddress,string fun
178 );
179 bytes32 constant EXCHANGE_DETAILS_TYPEHASH = keccak256("MetaTxExchangeDetails(uint256
180 bytes32 constant META_TX_EXCHANGE_TYPEHASH = keccak256(
181     "MetaTxExchange(uint256 nonce,address from,address contractAddress,string function
182 );
183 bytes32 constant FUND_DETAILS_TYPEHASH = keccak256(
184     "MetaTxFundDetails(uint256 entityId,address[] tokenList,uint256[] tokenAmounts)"
185 );
186 bytes32 constant META_TX_FUNDS_TYPEHASH = keccak256(
187     "MetaTxFund(uint256 nonce,address from,address contractAddress,string functionName
188 );
189 bytes32 constant DISPUTE_RESOLUTION_DETAILS_TYPEHASH = keccak256(
190     "MetaTxDisputeResolutionDetails(uint256 exchangeId,uint256 buyerPercent,bytes32 si
191 );
192 bytes32 constant META_TX_DISPUTE_RESOLUTIONS_TYPEHASH = keccak256(
193     "MetaTxDisputeResolution(uint256 nonce,address from,address contractAddress,string
194 );
```

### Recommendation:

We advise them to be corrected as they may lead to unexpected signed payloads for integrators.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

After evaluation of supplemental material provided to us by the Boson Protocol team we evaluated that their definitions of the struct types conform to the EIP-712 standard and as such this exhibit can be considered nullified.

# BosonVoucher Manual Review Findings

## BVR-01M: Inexistent Prevention of Self-Transfer

| Type | Severity | Location |
| --- | --- | --- |
| Logical Fault | Unknown | BosonVoucher.sol:L143 |

### Description:

The `_beforeTokenTransfer` hook executes sensitive `onVoucherTransferred` functionality that should be solely executed when dealing with different `from` and `to` addresses.

### Example:

contracts/protocol/clients/voucher/BosonVoucher.sol

SOL

```sol
137 function _beforeTokenTransfer(
138     address from,
139     address to,
140     uint256 tokenId
141 ) internal override {
142     // Only act when transferring, not minting or burning
143     if (from != address(0) && to != address(0)) {
144         onVoucherTransferred(tokenId, payable(to));
145     }
146 }
```

### Recommendation:

We advise the `if` conditional to additionally validate that `from` is not equal to `to`, preventing self-transfers from executing the `onVoucherTransferred` hooks and affecting future functionality of the protocol.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

An additional check preventing the execution of `onVoucherTransferred` when `from` is equal to `to` was included in the codebase thus alleviating this exhibit in full.

# BundleBase Manual Review Findings

## BBS-01M: Inexistent Validation of Valid Bundle Creation

| Type | Severity | Location |
| --- | --- | --- |
| Logical Fault | Minor | BundleBase.sol:L45, L48 |

### Description:

The `createBundleInternal` function does not properly validate that either `offerIds.length` or `twinIds.length` is non-zero, permitting a bundle to be created with zero offer IDs and zero twin IDs.

### Impact:

Incorrectly created bundles can lead to misbehaviours across the greater Boson Protocol ecosystem.

### Example:

contracts/protocol/bases/BundleBase.sol

SOL

```sol
36  function createBundleInternal(Bundle memory _bundle) internal {
37      // get message sender
38      address sender = msgSender();
39
40      // get seller id, make sure it exists and store it to incoming struct
41      (bool exists, uint256 sellerId) = getSellerIdByOperator(sender);
42      require(exists, NOT_OPERATOR);
43
44      // limit maximum number of offers to avoid running into block gas limit in a loop
45      require(_bundle.offerIds.length <= protocolLimits().maxOffersPerBundle, TOO_MANY_O
46
47      // limit maximum number of twins to avoid running into block gas limit in a loop
48      require(_bundle.twinIds.length <= protocolLimits().maxTwinsPerBundle, TOO_MANY_TWI
49
50      // Get the next bundle and increment the counter
51      uint256 bundleId = protocolCounters().nextBundleId++;
52      // Sum of offers quantity available
53      uint256 offersTotalQuantityAvailable;
54
```

```
55      for (uint256 i = 0; i < _bundle.offerIds.length; i++) {
56          uint256 offerId = _bundle.offerIds[i];
57
58          // Calculate bundle offers total quantity available.
59          offersTotalQuantityAvailable = calculateOffersTotalQuantity(offersTotalQuantit
60
61          (bool bundleByOfferExists, ) = fetchBundleIdByOffer(offerId);
62          require(!bundleByOfferExists, BUNDLE_OFFER_MUST_BE_UNIQUE);
63
64          (bool exchangeIdsForOfferExists, ) = getExchangeIdsByOffer(offerId);
65          // make sure exchange does not already exist for this offer id.
66          require(!exchangeIdsForOfferExists, EXCHANGE_FOR_OFFER_EXISTS);
67
68          // Add to bundleIdByOffer mapping
69          protocolLookups().bundleIdByOffer[offerId] = bundleId;
70      }
71
72      for (uint256 i = 0; i < _bundle.twinIds.length; i++) {
73          uint256 twinId = _bundle.twinIds[i];
74
75          // A twin can't belong to multiple bundles
76          (bool bundleForTwinExist, ) = fetchBundleIdByTwin(twinId);
77          require(!bundleForTwinExist, BUNDLE_TWIN_MUST_BE_UNIQUE);
78
79          if (_bundle.offerIds.length > 0) {
80              bundleSupplyChecks(offersTotalQuantityAvailable, twinId);
81          }
82
83          // Push to bundleIdsByTwin mapping
84          protocolLookups().bundleIdByTwin[_bundle.twinIds[i]] = bundleId;
85      }
86
87      // Get storage location for bundle
88      (, Bundle storage bundle) = fetchBundle(bundleId);
89
90      // Set bundle props individually since memory structs can't be copied to storage
91      bundle.id = _bundle.id = bundleId;
92      bundle.sellerId = _bundle.sellerId = sellerId;
93      bundle.offerIds = _bundle.offerIds;
94      bundle.twinIds = _bundle.twinIds;
95
96      // Notify watchers of state change
97      emit BundleCreated(bundleId, sellerId, _bundle, sender);
98  }
```

**Recommendation:**

We advise this trait of the system to be re-evaluated as it currently permits incorrect bundle creation.

**Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):**

The bundle creation process now mandates that at least one twin and one offer ID has been specified when a bundle is created thus alleviating this exhibit.

# ClientLib Manual Review Findings

## CLB-01M: Potentially Discrepant Access Control

| Type | Severity | Location |
| --- | --- | --- |
| **Standard Conformity** | Unknown | ClientLib.sol:L52-L55 |

### Description:

The `ClientLib` implementation is aware of both the `accessController` address and the `protocolDiamond` address, however, when applying access control the `accessController` locally held is queried instead of the `protocolDiamond` access control implementation which may differ.

### Example:

contracts/protocol/libs/ClientLib.sol

SOL

```sol
16  struct ProxyStorage {
17      // The AccessController address
18      IAccessControlUpgradeable accessController;
19      // The ProtocolDiamond address
20      address protocolDiamond;
21      // The client implementation address
22      address implementation;
23  }
```

### Recommendation:

We advise either the access control of `protocolDiamond` to be queried or, should a dedicated `accessController` be required for the `ProxyStorage`, the code to be adequately documented as such.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The `accessController` entry was omitted from the `ProxyStorage` and now the `address` is properly looked up via the `getAccessControllerAddress` getter function exposed by the `ps.protocolDiamond`.

# ConfigHandlerFacet Manual Review Findings

## CHF-01M: Potentially Overly Centralized Configuration Control

| Type | Severity | Location |
| --- | --- | --- |
| Centralization Concern | Unknown | ConfigHandlerFacet.sol:L76-L79, L95-L98, L114-L117, L133-L136 |

### Description:

The referenced functions permit the `ADMIN` role holder of the system to arbitrarily set very sensitive contract components (such as the `token` address of the protocol).

### Example:

contracts/protocol/facets/ConfigHandlerFacet.sol

SOL

```sol
/**
 * @notice Sets the address of the Boson Protocol token contract.
 *
 * Emits a TokenAddressChanged event.
 *
 * @param _tokenAddress - the address of the token contract
 */
function setTokenAddress(address payable _tokenAddress) public override onlyRole(ADMIN
    protocolAddresses().token = _tokenAddress;
    emit TokenAddressChanged(_tokenAddress, msgSender());
}
```

### Recommendation:

We advise this to be revised and very sensitive configurational variables to only be settable once during the contract's lifetime to avoid centralization risks.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The Boson Protocol team has opted to retain the current centralization structure in place after evaluating the exhibit given that they wish to apply a progressive decentralization plan to the protocol by utilizing a multi-

signature wallet for the initial `ADMIN`-role activities with a plan to completely abolish control of the account once the protocol has sufficiently matured. As a result, we consider this exhibit as addressed **based on the principle that the Boson Protocol team will act in good faith** with regards to the protocol when administering `ADMIN`-role related activities as no adequate decentralized solution currently exists for the protocol.

## CHF-02M: Inexistent Validation of Numeric Configuration

| Type | Severity | Location |
| --- | --- | --- |
| **Input Sanitization** | Minor | ConfigHandlerFacet.sol:L205, L224, L243, L262, L281, L300, L324, L348, L432, L517, L536 |

### Description:

The linked numeric configurational parameters are not properly sanitized which could lead to significant protocol consequences, such as the `maxTokensPerWithdrawal` value being set to `0` which would in turn not allow tokens to be withdrawn from the system.

### Impact:

A zero value for most of these parameters would lead to misbehaviours or inoperable system components such as the illustrated maximum tokens per withdrawal preventing any withdrawal to be performed in the `FundsHandlerFacet` implementation.

### Example:

contracts/protocol/facets/ConfigHandlerFacet.sol

SOL

```
274 /**
275  * @notice Sets the maximum numbers of tokens that can be withdrawn in a single transa
276  *
277  * Emits a mMxTokensPerWithdrawalChanged event.
278  *
279  * @param _maxTokensPerWithdrawal - the maximum length of token list when calling {Fun
280  */
281 function setMaxTokensPerWithdrawal(uint16 _maxTokensPerWithdrawal) public override onl
282     protocolLimits().maxTokensPerWithdrawal = _maxTokensPerWithdrawal;
283     emit MaxTokensPerWithdrawalChanged(_maxTokensPerWithdrawal, msgSender());
284 }
```

### Recommendation:

We advise a non-zero check to be imposed on all referenced input arguments to ensure the contract is configured properly at all times.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

A non-zero check has been introduced across all referenced setter functions ensuring that the contract is properly configured in relation to its numerical values at all times.

# DisputeBase Manual Review Findings

## DBE-01M: Potentially Misleading Validation Comments

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Unknown | DisputeBase.sol:L21 |

### Description:

The linked comment states that the `raiseDisputeInternal` function validates the state of the exchange is in a redeemed state, however, this type of validation is performed either directly or indirectly wherever the function is invoked.

### Example:

contracts/protocol/bases/DisputeBase.sol

SOL

```sol
15  /**
16   * @dev Raise a dispute
17   *
18   * Reverts if:
19   * - caller does not hold a voucher for the given exchange id
20   * - exchange does not exist
21   * - exchange is not in a redeemed state
22   * - fulfillment period has elapsed already
23   *
24   * @param _exchange - the exchange
25   * @param _voucher - the associated voucher
26   * @param _sellerId - the seller id
27   */
28  function raiseDisputeInternal(
29      Exchange storage _exchange,
30      Voucher storage _voucher,
31      uint256 _sellerId
32  ) internal {
33      // Make sure the fulfillment period has elapsed
34      uint256 elapsed = block.timestamp - _voucher.redeemedDate;
35      require(elapsed < fetchOfferDurations(_exchange.offerId).fulfillmentPeriod, FULFIL
36
```

```
37      // Make sure the caller is buyer associated with the exchange
38      checkBuyer(_exchange.buyerId);
39
40      // Set the exchange state to disputed
41      _exchange.state = ExchangeState.Disputed;
42
43      // Fetch the dispute and dispute dates
44      (, Dispute storage dispute, DisputeDates storage disputeDates) = fetchDispute(_exc
45
46      // Set the initial values
47      dispute.exchangeId = _exchange.id;
48      dispute.state = DisputeState.Resolving;
49
50      // Update the disputeDates
51      disputeDates.disputed = block.timestamp;
52      disputeDates.timeout = block.timestamp + fetchOfferDurations(_exchange.offerId).re
53
54      // Notify watchers of state change
55      emit DisputeRaised(_exchange.id, _exchange.buyerId, _sellerId, msgSender());
56  }
```

### Recommendation:

We advise either the validation to be relocated to the function or the comment to be omitted as the validation statement is incorrect.

### Alleviation (f62f5f26c278f8bbe55600c7bb344d5941da8787):

The incorrect comment was omitted in the latest iteration of the report as advised.

# EIP712Lib Manual Review Findings

## EIP-01M: Inexistent Validation of Meta-TX Sender Validity

| Type | Severity | Location |
| --- | --- | --- |
| **Language Specific** | Unknown | EIP712Lib.sol:L102 |

### Description:

The sender of a particular transaction should at all times not be equal to the zero-address as a standard throughout the ecosystem due to the said address' special purpose.

### Example:

contracts/protocol/libs/EIP712Lib.sol

SOL

```sol
94  /**
95   * @notice Returns the current sender address.
96   */
97  function msgSender() internal view returns (address) {
98      bool isItAMetaTransaction = ProtocolLib.protocolMetaTxInfo().isMetaTransaction;
99
100     // Get sender from the storage if this is a meta transaction
101     if (isItAMetaTransaction) {
102         return getCurrentSenderAddress();
103     } else {
104         return msg.sender;
105     }
106 }
```

### Recommendation:

We advise such a check to be imposed here by mandating that `getCurrentSenderAddress` is non-zero as a case of `isItAMetaTransaction` and `getCurrentSenderAddress` being zero should be considered invalid.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The result of `getCurrentSenderAddress` is now properly validated to be non-zero in conformance with the EVM ecosystem's standards.

# EIP-02M: Improper Domain Separator Retrieval

| Type | Severity | Location |
| --- | --- | --- |
| **Language Specific** | Minor | EIP712Lib.sol:L67-L72 |

### Description:

The `EIP712Lib` contract contains functions that dynamically construct the currently valid domain separator (i.e. `domainSeparator`) yet when `getDomainSeparator` is invoked the last value stored to `domainSeparator` is retrieved instead. As evidenced in the `ConfigHandlerFacet` contract that initializes this value, the `domainSeparator` is assigned to once thus permitting cross-chain replay attacks to occur, a trait especially relevant with the upcoming Ethereum hard fork.

### Impact:

As the `domainSeparator` currently remains the same regardless of whether the blockchain the contract was deployed in forked, it is possible for a signed **EIP-712** payload that is submitted on the original chain to be replayed on the forked chain improperly.

### Example:

contracts/protocol/libs/EIP712Lib.sol

SOL

```sol
67  /**
68   * @notice Get the domain separator.
69   */
70  function getDomainSeparator() private view returns (bytes32) {
71      return ProtocolLib.protocolMetaTxInfo().domainSeparator;
72  }
```

### Recommendation:

We advise a caching system to be used instead similarly to the draft **EIP-712** implementation by OpenZeppelin whereby the domain separator is re-calculated on a need-to basis by comparing the current `block.chainid` and the one that the original `domainSeparator` was calculated in, allowing the separator to be re-calculated in case the chain IDs do not match (i.e. due to the PoW chain of Ethereum being used instead of the PoS one).

instead of the PoS one).

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

A proxy-compliant caching system is now utilized that builds the domain separator on a need-to basis depending on whether the chain ID identified during the deployment of the contract has changed in the current execution environment. As a result, we consider this exhibit alleviated in full.

# EIP-03M: Insecure Elliptic Curve Recovery Mechanism

| Type | Severity | Location |
| --- | --- | --- |
| Language Specific | Medium | EIP712Lib.sol:L59, L60 |

### Description:

The `ecrecover` function is a low-level cryptographic function that should be utilized after appropriate sanitizations have been enforced on its arguments, namely on the `s` and `v` values. This is due to the inherent trait of the curve to be symmetrical on the x-axis and thus permitting signatures to be replayed with the same `x` value (`r`) but a different `y` value (`s`).

### Impact:

Should the payload being verified by the signature rely on differentiation based on the `s` or `v` arguments, it will be possible to replay the signature for the same data validly and acquire authorization twice.

### Example:

contracts/protocol/libs/EIP712Lib.sol

SOL

```sol
function verify(
    address _user,
    bytes32 _hashedMetaTx,
    bytes32 _sigR,
    bytes32 _sigS,
    uint8 _sigV
) internal view returns (bool) {
    address signer = ecrecover(toTypedMessageHash(_hashedMetaTx), _sigV, _sigR, _sigS)
    require(signer != address(0), INVALID_SIGNATURE);
    return signer == _user;
}
```

### Recommendation:

We advise them to be sanitized by ensuring that `v` is equal to either `27` or `28` (`v ∈ {27, 28}`) and to ensure that `s` is existent in the lower half order of the elliptic curve (`0 < s < secp256k1n ÷ 2 + 1`) by ensuring it is less than `0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A1`. A

ensuring it is less than `0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A1`. A

reference implementation of those checks can be observed in the **ECDSA** library of OpenZeppelin and the rationale behind those restrictions exists within **Appendix F of the Yellow Paper**.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The `_sigS` and `_sigV` values are now adequately validated within the `verify` function according to the EVM yellow-paper standards.

# ExchangeHandlerFacet Manual Review Findings

## EHF-01M: Improper Invocation of EIP-20 `transfer`

| Type | Severity | Location |
| --- | --- | --- |
| **Standard Conformity** | Medium | ExchangeHandlerFacet.sol:L649-L656 |

### Description:

The linked statement does not properly validate the returned `bool` of the **EIP-20** standard `transfer` function. As the **standard dictates**, callers **must not** assume that `false` is never returned.

### Impact:

If the code mandates that the returned `bool` is `true`, this will cause incompatibility with tokens such as USDT / Tether as no such `bool` is returned to be evaluated causing the check to fail at all times. On the other hand, if the token utilized can return a `false` value under certain conditions but the code does not validate it, the contract itself can be compromised as having received / sent funds that it never did.

### Example:

contracts/protocol/facets/ExchangeHandlerFacet.sol

SOL

```sol
649 (success, result) = twin.tokenAddress.call(
650     abi.encodeWithSignature(
651         "transferFrom(address,address,uint256)",
652         seller.operator,
653         sender,
654         twin.amount
655     )
656 );
```

### Recommendation:

Since not all standardized tokens are **EIP-20** compliant (such as Tether / USDT), we advise a safe wrapper library to be utilized instead such as `SafeERC20` by OpenZeppelin to opportunistically validate the returned `bool` only if it exists.

### Alleviation (44009967e4):

The `transferFrom` function invoked in the referenced statement is still performed using its unsafe counterpart whilst a `bool` may never be yielded by the specified `tokenAddress`. We advise the Boson Protocol team to revisit this exhibit.

### Alleviation (6dae5d2602):

After re-visiting this exhibit, the Boson Protocol team highlighted that the `success` and `result` variable are both utilized in the code similarly to how the `safeTransferFrom` function of OpenZeppelin behaves and as such the code works as intended. In light of this, we consider this exhibit nullified as it never presented an issue.

# FundsHandlerFacet Manual Review Findings

## FHF-01M: Inexistent Validation of Token Type

| Type | Severity | Location |
| --- | --- | --- |
| **Logical Fault** | Minor | FundsHandlerFacet.sol:L43-L67 |

### Description:

The `depositFunds` function permits an arbitrary caller to deposit funds available for a particular `_sellerId` with no form of validation on the caller or input arguments, thereby permitting elaborate phishing attacks to occur whereby a user invokes `depositFunds` with a malicious token to a particular user's account and when the target user invokes `withdrawFunds` with the automatic method on, the malicious token balance is automatically withdrawn to the user.

### Impact:

By depositing funds to a particular seller prior to f.e. their withdrawal transaction being processed, it is possible to "inject" a malicious token balance the user will withdraw seemingly from the Boson Protocol which will make the funds seem "trustworthy" and potential "bonuses" provided by Boson. This in turn will render phishing attacks easier to execute for unsuspecting users.

### Example:

contracts/protocol/facets/FundsHandlerFacet.sol

SOL

```sol
43  function depositFunds(
44      uint256 _sellerId,
45      address _tokenAddress,
46      uint256 _amount
47  ) external payable override fundsNotPaused nonReentrant {
48      //Check Seller exists in sellers mapping
49      (bool exists, , ) = fetchSeller(_sellerId);
50
51      //Seller must exist
52      require(exists, NO_SUCH_SELLER);
53
```

```
54      if (msg.value != 0) {
55          // receiving native currency
56          require(_tokenAddress == address(0), NATIVE_WRONG_ADDRESS);
57          require(_amount == msg.value, NATIVE_WRONG_AMOUNT);
58      } else {
59          // transfer tokens from the caller
60          FundsLib.transferFundsToProtocol(_tokenAddress, _amount);
61      }
62
63      // increase available funds
64      FundsLib.increaseAvailableFunds(_sellerId, _tokenAddress, _amount);
65
66      emit FundsDeposited(_sellerId, msgSender(), _tokenAddress, _amount);
67  }
```

### Recommendation:

We advise the function to either permit depositing to existing tokens in the token list of a seller or to utilize some other form of validation method as currently it will turn into a hot-spot for phishing attacks as funds transferred from the Boson Protocol to the user would appear more "trustworthy" than a phishing attempt by directly transferring funds to the user.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The Boson Protocol team opted to not remediate the attack vector described at this time given that this relates to user-related activities and has no impact to the protocol. We would like to note that it is possible to completely halt the automatic withdrawal feature (which some external integrators may rely on) by injecting a malicious token that fails on all transfers and as such would cause otherwise expected protocol behaviour to fail. As a result, we retain the `minor` severity level and set this finding as acknowledged.

# GroupBase Manual Review Findings

## GBE-01M: Inexistent Validation of Group Entries

| Type | Severity | Location |
| --- | --- | --- |
| **Input Sanitization** | Minor | GroupBase.sol:L145 |

### Description:

The `_offerIds` is not validated as a non-empty array of offers.

### Impact:

Empty groups could cause certain code paths to fail instead of being executed properly.

### Example:

contracts/protocol/bases/GroupBase.sol

SOL

```sol
141 function addOffersToGroupInternal(uint256 _groupId, uint256[] memory _offerIds) intern
142     // check if group can be updated
143     (uint256 sellerId, Group storage group) = preUpdateChecks(_groupId, _offerIds);
144
145     for (uint256 i = 0; i < _offerIds.length; i++) {
```

### Recommendation:

We advise such validation to be imposed as otherwise empty groups may be created that can lead to unintended consequences.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

After discussion with the Boson Protocol team, we assessed that the check recommended is already applied indirectly by the `preUpdateChecks` function invoked thus nullifying this exhibit.

## GBE-02M: Insufficient Evaluation of `maxOffersPerGroup`

| Type | Severity | Location |
| --- | --- | --- |
| **Input Sanitization** | Minor | GroupBase.sol:L196 |

### Description:

The `maxOffersPerGroup` limitation can be bypassed as the `preUpdateChecks` function does not take into account existing checks within the group thus permitting the extension of a group's offers indefinitely.

### Impact:

The `maxOffersPerGroup` limitation is not properly upheld by the contract which can cause denial-of-service attacks on functions utilizing groups.

### Example:

contracts/protocol/bases/GroupBase.sol

SOL

```sol
171 /**
172  * @dev Before performing an update, make sure update can be done
173  * and return seller id and group storage pointer for further use
174  *
175  * Reverts if:
176  *
177  * - caller is not the seller
178  * - offer ids is an empty list
179  * - number of offers exceeds maximum allowed number per group
180  * - group does not exist
181  *
182  * @param _groupId  - the id of the group to be updated
183  * @param _offerIds - array of offer ids to be removed to the group
184  * @return sellerId  - the seller Id
185  * @return group - the group details
186  */
187 function preUpdateChecks(uint256 _groupId, uint256[] memory _offerIds)
188     internal
189     view
190     returns (uint256 sellerId, Group storage group)
```

```
191 {
192     // make sure that at least something will be updated
193     require(_offerIds.length != 0, NOTHING_UPDATED);
194
195     // limit maximum number of offers to avoid running into block gas limit in a loop
196     require(_offerIds.length <= protocolLimits().maxOffersPerGroup, TOO_MANY_OFFERS);
```

### Recommendation:

We advise the `preUpdateChecks` function to also sum the total number of offers already existing in the group with the `_offerIds.length` value to properly enforce the `maxOffersPerGroup` limitation.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The `maxOffersPerGroup` limitation is now properly applied at the `addOffersToGroupInternal` level thus alleviating this exhibit in full.

# MetaTransactionsHandlerFacet Manual Review Findings

## MTH-01M: Account Agnostic Nonce System

| Type | Severity | Location |
| --- | --- | --- |
| Logical Fault | Medium | MetaTransactionsHandlerFacet.sol:L173, L218 |

### Description:

The cryptographic system of the `MetaTransactionsHandlerFacet` does not utilize an account-specific `nonce` thus paving way to race conditions that allow selectively cancelling meta transactions at will.

### Impact:

It is possible to hi-jack meta-transaction execution flows that can have varying degrees of impact ranging from locked funds to complete transaction workflow failures.

### Example:

contracts/protocol/facets/MetaTransactionsHandlerFacet.sol

SOL

```sol
168 function validateTx(
169     string calldata _functionName,
170     bytes calldata _functionSignature,
171     uint256 _nonce
172 ) internal view {
173     require(!protocolMetaTxInfo().usedNonce[_nonce], NONCE_USED_ALREADY);
174 
175     bytes4 destinationFunctionSig = convertBytesToBytes4(_functionSignature);
176     require(destinationFunctionSig != msg.sig, INVALID_FUNCTION_SIGNATURE);
177 
178     bytes4 functionNameSig = bytes4(keccak256(abi.encodePacked(_functionName)));
179     require(destinationFunctionSig == functionNameSig, INVALID_FUNCTION_NAME);
180 }
```

### Recommendation:

We advise an account-based `nonce` to be utilized instead that ensures the transactions are executed in the sequence they are meant to and that the sequence cannot be hijacked as currently a malicious user can detect a transaction that they wish to cancel and submit their own no-op meta-transaction with the same `nonce` thus invalidating it.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The contract's code has been updated to introduce an additional `address` based key to the `usedNonce` mapping thus ensuring that the nonce system is bound to each account and cannot be hi-jacked by external account actions.

# OfferBase Manual Review Findings

## OBE-01M: Inexistent Validation of Proper Voucher Redemption Setup

| Type | Severity | Location |
| --- | --- | --- |
| **Input Sanitization** | Minor | OfferBase.sol:L123-L127 |

### Description:

The `voucherRedeemableFrom` and `voucherRedeemableUntil` values can be one unit in difference and the `voucherRedeemableUntil` can be equal to `validUntil` in which cases the redemption window is inadequate and inoperable (i.e. a purchase made at `validUntil` will have no time to be redeemed as well as a window of `1` second will not be usable).

### Impact:

An improperly set-up voucher system could allow vouchers to be purchased that ultimately will not be redeemable and thus cause automatic forfeiture of funds for the buyers improperly.

### Example:

contracts/protocol/bases/OfferBase.sol

SOL

```
123 if (_offerDates.voucherRedeemableUntil > 0) {
124     require(_offerDurations.voucherValid == 0, AMBIGUOUS_VOUCHER_EXPIRY);
125     require(_offerDates.voucherRedeemableFrom < _offerDates.voucherRedeemableUntil, RE
126     require(_offerDates.voucherRedeemableUntil >= _offerDates.validUntil, REDEMPTION_P
127 } else {
```

### Recommendation:

We advise proper validation to be imposed on the voucher redemption system by setting a group of "minimum" values that act as a "grace period" for voucher redemption ensuring that redemptions can be timely executed.

**Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):**

After evaluation of the exhibit, the Boson Protocol team decided to retain the current code in place and to not introduce any validation checks as the wish to ensure all partners and integrators are not significantly impacted functionally and plan to potentially remediate in a post-v2 upgrade. As a result, we consider this exhibit acknowledged.

# OfferHandlerFacet Manual Review Findings

## OHF-01M: Inexistent Re-Entrancy Protection

| Type | Severity | Location |
| --- | --- | --- |
| Language Specific | Minor | OfferHandlerFacet.sol:L107 |

### Description:

The `createOfferBatch` function does not provide re-entrancy protection in contrast to the `createOffer` function.

### Impact:

A re-entrancy during the creation of an offer can cause a corrupted order to be stored and potentially exploited via another, `nonReentrant` protected pathway of the Boson Protocol.

### Example:

contracts/protocol/facets/OfferHandlerFacet.sol

SOL

```sol
function createOffer(
    Offer memory _offer,
    OfferDates calldata _offerDates,
    OfferDurations calldata _offerDurations,
    uint256 _disputeResolverId,
    uint256 _agentId
) external override offersNotPaused nonReentrant {
    createOfferInternal(_offer, _offerDates, _offerDurations, _disputeResolverId, _age
}

/**
 * @notice Creates a batch of offers.
 *
 * Emits an OfferCreated event for every offer if successful.
 *
 * Reverts if:
 * - The offers region of protocol is paused
 * - Number of offers exceeds maximum allowed number per batch
```

```
73   * - Number of elements in offers, offerDates and offerDurations do not match
74   * - for any offer:
75   *   - Caller is not an operator
76   *   - Valid from date is greater than valid until date
77   *   - Valid until date is not in the future
78   *   - Both voucher expiration date and voucher expiraton period are defined
79   *   - Neither of voucher expiration date and voucher expiraton period are defined
80   *   - Voucher redeemable period is fixed, but it ends before it starts
81   *   - Voucher redeemable period is fixed, but it ends before offer expires
82   *   - Fulfillment period is set to zero
83   *   - Resolution period is set to zero
84   *   - Voided is set to true
85   *   - Available quantity is set to zero
86   *   - Dispute resolver wallet is not registered, except for absolute zero offers with
87   *   - Dispute resolver is not active, except for absolute zero offers with unspecifie
88   *   - Seller is not on dispute resolver's seller allow list
89   *   - Dispute resolver does not accept fees in the exchange token
90   *   - Buyer cancel penalty is greater than price
91   * - When agent ids are non zero:
92   *   - If Agent does not exist
93   *   - If the sum of Agent fee amount and protocol fee amount is greater than the offe
94   *
95   * @param _offers - the array of fully populated Offer structs with offer id set to 0x
96   * @param _offerDates - the array of fully populated offer dates structs
97   * @param _offerDurations - the array of fully populated offer durations structs
98   * @param _disputeResolverIds - the array of ids of chosen dispute resolvers (can be 0
99   * @param _agentIds - the array of ids of agents
100  */
101 function createOfferBatch(
102     Offer[] calldata _offers,
103     OfferDates[] calldata _offerDates,
104     OfferDurations[] calldata _offerDurations,
105     uint256[] calldata _disputeResolverIds,
106     uint256[] calldata _agentIds
107 ) external override offersNotPaused {
```

### Recommendation:

We advise the `nonReentrant` modifier to properly be applied to the function as it interacts with unknown tokens which may cause the offer creation workflow to be hijacked mid-creation.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The `nonReentrant` modifier has been properly introduced to the `createOfferBatch` function as advised.

# ProtocolBase Manual Review Findings

## PBS-01M: Potentially Unsafe Existence Check

| Type | Severity | Location |
| --- | --- | --- |
| **Language Specific** | Unknown | ProtocolBase.sol:L720 |

### Description:

The linked existence check utilizes a fixed `enum` value instead of evaluating the actual numeric representation of the `enum` being equal to `0`.

### Example:

contracts/protocol/bases/ProtocolBase.sol

SOL

```sol
704 /**
705  * @notice Fetches a condition from storage by exchange id
706  *
707  * @param _exchangeId - the id of the exchange
708  * @return exists - whether one condition exists for the exchange
709  * @return condition - the condition. See {BosonTypes.Condition}
710  */
711 function fetchConditionByExchange(uint256 _exchangeId)
712     internal
713     view
714     returns (bool exists, Condition storage condition)
715 {
716     // Get the condition slot
717     condition = protocolLookups().exchangeCondition[_exchangeId];
718
719     // Determine existence
720     exists = (_exchangeId > 0 && condition.method != EvaluationMethod.None);
721 }
```

### Recommendation:

We advise the actual numeric representation to be utilized that correctly depicts its "unset" status as should

the `enum` declaration change in order the `EvaluationMethod.None` could be depicting a non-zero state thus invalidating the existence check.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The Boson Protocol team stated that they will retain the current code in place and will simply ensure via test suites that the `None` evaluation method always exist in the first `enum` slot as that is its intended behaviour. We consider this exhibit as addressed **based on the assumption that the Boson Protocol team will guarantee the `None` value's validity** in all iterations of the codebase.

# SellerBase Manual Review Findings

## SBE-01M: Potential of Race Condition in Seller Creation

| Type | Severity | Location |
| --- | --- | --- |
| Logical Fault | Medium | SellerBase.sol:L32-L86 |

### Description:

The `createSellerInternal` function as well as where it is called apply no access control to the caller. Given that the seller creation system strictly associates user types with a single seller ID, it is possible for a race condition to manifest by detecting the creation of a seller account and adjusting the `treasury` address to a different one causing an unsuspecting user to never receive the funds they are owed for a sale.

### Example:

contracts/protocol/bases/SellerBase.sol

SOL

```
32  function createSellerInternal(
33      Seller memory _seller,
34      AuthToken calldata _authToken,
35      VoucherInitValues calldata _voucherInitValues
36  ) internal {
37      //Check active is not set to false
38      require(_seller.active, MUST_BE_ACTIVE);
39
40      //Admin address or AuthToken data must be present. A seller can have one or the ot
41      require(
42          (_seller.admin == address(0) && _authToken.tokenType != AuthTokenType.None) ||
43              (_seller.admin != address(0) && _authToken.tokenType == AuthTokenType.None
44          ADMIN_OR_AUTH_TOKEN
45      );
46
47      //Check that the addresses are unique to one seller Id, accross all roles. These a
48      require(
49          protocolLookups().sellerIdByOperator[_seller.operator] == 0 &&
50              protocolLookups().sellerIdByOperator[_seller.clerk] == 0 &&
51              protocolLookups().sellerIdByAdmin[_seller.operator] == 0 &&
52              protocolLookups().sellerIdByAdmin[_seller.clerk] == 0 &&
```

```
            protocolLookups().sellerIdByClerk[_seller.operator] == 0 &&
            protocolLookups().sellerIdByClerk[_seller.clerk] == 0,
        SELLER_ADDRESS_MUST_BE_UNIQUE
    );

    //Do other uniqueness checks based on auth type
    if (_seller.admin == address(0)) {
        //Check that auth token is unique to this seller
        require(
            protocolLookups().sellerIdByAuthToken[_authToken.tokenType][_authToken.tok
            AUTH_TOKEN_MUST_BE_UNIQUE
        );
    } else {
        //check that the admin address is unique to one seller Id, accross all roles
        require(
            protocolLookups().sellerIdByOperator[_seller.admin] == 0 &&
                protocolLookups().sellerIdByAdmin[_seller.admin] == 0 &&
                protocolLookups().sellerIdByClerk[_seller.admin] == 0,
            SELLER_ADDRESS_MUST_BE_UNIQUE
        );
    }

    // Get the next account Id and increment the counter
    uint256 sellerId = protocolCounters().nextAccountId++;
    _seller.id = sellerId;
    storeSeller(_seller, _authToken);

    // create clone and store its address cloneAddress
    address voucherCloneAddress = cloneBosonVoucher(sellerId, _seller.operator, _vouch
    protocolLookups().cloneAddress[sellerId] = voucherCloneAddress;

    // Notify watchers of state change
    emit SellerCreated(sellerId, _seller, voucherCloneAddress, _authToken, msgSender()
}
```

### Recommendation:

We advise the seller creation flow to apply some form of access control to the caller or sanitization of the input arguments to ensure that an account cannot be created for other users and thus disallowing this type of fraud attempt from ever being feasible on-chain.

### Alleviation (68ebb15f14b12c7e64c233ff00363398a127a71e):

The `createSellerInternal` function now mandates that the `_seller.operator` and `_seller.clerk` values (as well as, conditionally, the `_seller.admin` value) all represent the current transaction sender thus

preventing accounts to be created for other users. To ensure the system still permits an account to be updated in relation to those values, an opt-in system has been introduced to the `SellerHandlerFacet` implementation that treats updates as "pending" until the recipient of the role "opts-in" to the upgrade manually, ensuring accounts and responsibilities cannot be offloaded to other addresses without confirmation of receipt. As a result of this workflow adjustment, we consider this exhibit alleviated in full.

# BosonConstants Code Style Findings

## BCS-01C: Generic Typographic Mistakes

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | BosonConstants.sol:L54, L59, L80 |

### Description:

The referenced lines contain typographical mistakes or generic documentational errors (i.e. copy-paste) that should be corrected.

### Example:

contracts/domain/BosonConstants.sol

```sol
54  string constant INVALID_AMOUNT_DISPUTE_RESOLVER_FEES = "Dispute resolver fees are not
```

### Recommendation:

We advise them to be corrected enhancing the legibility of the codebase.

### Alleviation (6dae5d260243cdab934f327402ee044ada337fa9):

All typographic mistakes pointed out in the exhibit as well as numerous others that the Boson Protocol team identified by running a spell-checker across the codebase have been adequately dealt with.

# BosonTypes Code Style Findings

## BTS-01C: Potentially Harmful Default Enum State

| Type | Severity | Location |
| --- | --- | --- |
| **Language Specific** | Informational | BosonTypes.sol:L11-L25, L33-L40, L42-L49 |

### Description:

The linked `enum` declaration has a default state (the first "state" defined) that does not represent "zero" and instead alludes to an active state.

### Example:

contracts/domain/BosonTypes.sol

SOL

```sol
11  enum PausableRegion {
12      Offers,
13      Twins,
14      Bundles,
15      Groups,
16      Sellers,
17      Buyers,
18      DisputeResolvers,
19      Agents,
20      Exchanges,
21      Disputes,
22      Funds,
23      Orchestration,
24      MetaTransaction
25  }
```

### Recommendation:

We advise a default state to be defined (i.e. `Undefined` or `None`) to ensure that uninitialized variables such as `mapping` entries do not represent an active state that should otherwise be attained by valid entities.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The Boson Protocol team evaluated this exhibit and opted to retain the current `enum` definition in place as the values are utilized as powers of two and thus behave as expected. As a result, we consider this exhibit nullified.

# BosonVoucher Code Style Findings

## BVR-01C: Hard-Coded Interface Type

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | BosonVoucher.sol:L98 |

### Description:

The `ERC2981` interface ID hard-coded in the referenced statement is ill-advised.

### Example:

contracts/protocol/clients/voucher/BosonVoucher.sol

SOL

```
89  * 0x2a55205a representes ERC2981 interface id
90   */
91  function supportsInterface(bytes4 interfaceId)
92      public
93      view
94      override(ERC721Upgradeable, IERC165Upgradeable)
95      returns (bool)
96  {
97      return (interfaceId == type(IBosonVoucher).interfaceId ||
98          interfaceId == 0x2a55205a ||
99          super.supportsInterface(interfaceId));
100 }
```

### Recommendation:

We advise the `ERC2981` interface to be imported in the codebase and the `interfaceId` specifier to be utilized for specifying the interface the contract is meant to support.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The `interfaceId` specifier of the `IERC2981Upgradeable` type is now correctly utilized in place of the literal optimizing the legibility of the codebase.

# BundleBase Code Style Findings

## BBS-01C: Repetitive Invocations of Getter Function

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | BundleBase.sol:L45, L48, L69, L84 |

### Description:

The linked statements repetitively invoke the same getter function whilst its return value remains unchanged.

### Example:

contracts/protocol/bases/BundleBase.sol

SOL

```sol
for (uint256 i = 0; i < _bundle.offerIds.length; i++) {
    uint256 offerId = _bundle.offerIds[i];

    // Calculate bundle offers total quantity available.
    offersTotalQuantityAvailable = calculateOffersTotalQuantity(offersTotalQuantityAva

    (bool bundleByOfferExists, ) = fetchBundleIdByOffer(offerId);
    require(!bundleByOfferExists, BUNDLE_OFFER_MUST_BE_UNIQUE);

    (bool exchangeIdsForOfferExists, ) = getExchangeIdsByOffer(offerId);
    // make sure exchange does not already exist for this offer id.
    require(!exchangeIdsForOfferExists, EXCHANGE_FOR_OFFER_EXISTS);

    // Add to bundleIdByOffer mapping
    protocolLookups().bundleIdByOffer[offerId] = bundleId;
}

for (uint256 i = 0; i < _bundle.twinIds.length; i++) {
    uint256 twinId = _bundle.twinIds[i];

    // A twin can't belong to multiple bundles
    (bool bundleForTwinExist, ) = fetchBundleIdByTwin(twinId);
    require(!bundleForTwinExist, BUNDLE_TWIN_MUST_BE_UNIQUE);

    if (_bundle.offerIds.length > 0) {
```

```
79      if (_bundle.offerIds.length > 0) {
80          bundleSupplyChecks(offersTotalQuantityAvailable, twinId);
81      }
82
83      // Push to bundleIdsByTwin mapping
84      protocolLookups().bundleIdByTwin[_bundle.twinIds[i]] = bundleId;
85  }
```

### Recommendation:

We advise the getter function to be invoked once and its result to be stored to a local variable that is consequently utilized optimizing the gas cost of the statements significantly.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The protocol limits and lookups are now correctly cached to local variable declarations thus optimizing the codebase's gas cost.

# BundleHandlerFacet Code Style Findings

## BHF-01C: Unused Code

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | BundleHandlerFacet.sol:L15-L18 |

### Description:

The `BundleUpdateAttribute` remains unused in the codebase.

### Example:

contracts/protocol/facets/BundleHandlerFacet.sol

SOL

```sol
15  enum BundleUpdateAttribute {
16      TWIN,
17      OFFER
18  }
```

### Recommendation:

We advise it to either be safely omitted or update functionality to be introduced to the codebase, either of which would be a sufficient alleviation of this exhibit.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The redundant `enum` declaration has been safely omitted from the codebase.

# DiamondLib Code Style Findings

## DLB-01C: Redundant Conditional Return Statement

| Type | Severity | Location |
| --- | --- | --- |
| **Gas Optimization** | Informational | DiamondLib.sol:L81 |

### Description:

The `|| false` conditional included in the `return` statement of `supportsInterface` is redundant as the `ds.supportedInterfaces[_interfaceId]` value can only be either `true` when explicitly set or `false` when explicitly set or unset.

### Example:

contracts/diamond/DiamondLib.sol

SOL

```sol
81  return ds.supportedInterfaces[_interfaceId] || false;
```

### Recommendation:

We advise the latter of the conditional to be omitted optimizing the gas cost of the function in all `false` cases.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The redundant conditional evaluation has been safely omitted from the codebase.

# DisputeBase Code Style Findings

## DBE-01C: Repetitive Invocations of Getter Function

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | DisputeBase.sol:L35, L52 |

### Description:

The linked statements repetitively invoke the same getter function whilst its return value remains unchanged.

### Example:

contracts/protocol/bases/DisputeBase.sol

SOL

```sol
35  require(elapsed < fetchOfferDurations(_exchange.offerId).fulfillmentPeriod, FULFILLMEN
36
37  // Make sure the caller is buyer associated with the exchange
38  checkBuyer(_exchange.buyerId);
39
40  // Set the exchange state to disputed
41  _exchange.state = ExchangeState.Disputed;
42
43  // Fetch the dispute and dispute dates
44  (, Dispute storage dispute, DisputeDates storage disputeDates) = fetchDispute(_exchang
45
46  // Set the initial values
47  dispute.exchangeId = _exchange.id;
48  dispute.state = DisputeState.Resolving;
49
50  // Update the disputeDates
51  disputeDates.disputed = block.timestamp;
52  disputeDates.timeout = block.timestamp + fetchOfferDurations(_exchange.offerId).resolu
```

### Recommendation:

We advise the getter function to be invoked once and its result to be stored to a local variable that is consequently utilized optimizing the gas cost of the statements significantly.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The offer durations are now correctly cached to local variable declarations thus optimizing the codebase's gas cost.

# EIP712Lib Code Style Findings

## EIP-01C: Deprecated Chain ID Retrieval Code

| Type | Severity | Location |
| --- | --- | --- |
| **Gas Optimization** | Informational | EIP712Lib.sol:L38-L40 |

### Description:

The linked `assembly` block is meant to retrieve the currently active chain ID via the `chainid` assembly instruction, however, the `block.chainid` globally available variable is meant to be used instead since Solidity `0.8.X`.

### Example:

contracts/protocol/libs/EIP712Lib.sol

SOL

```sol
32  /**
33   * @notice Get the chain id
34   *
35   * @return id - the chain id, 1 for Ethereum mainnet, > 1 for public testnets.
36   */
37  function getChainID() internal view returns (uint256 id) {
38      assembly {
39          id := chainid()
40      }
41  }
```

### Recommendation:

We advise the `block.chainid` variable to be yielded by the `getChainID` function or as an additional optimization step the `block.chainid` variable to be used in place of the `getChainID` function call thus optimizing the codebase's gas cost.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The `getChainID` function has been omitted entirely in favour of direct `block.chainid` statements thus

optimizing the codebase's legibility and gas cost.

# ExchangeHandlerFacet Code Style Findings

## EHF-01C: Multiple Top-Level Type Declarations

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | ExchangeHandlerFacet.sol:L14, L20, L29 |

### Description:

The `ExchangeHandlerFacet` file contains three top-level declarations (`Token`, `MultiToken` and `ExchangeHandlerFacet`).

### Example:

contracts/protocol/facets/ExchangeHandlerFacet.sol

SOL

```sol
14  interface Token {
15      function balanceOf(address account) external view returns (uint256); //ERC-721 and
16
17      function ownerOf(uint256 _tokenId) external view returns (address); //ERC-721
18  }
19
20  interface MultiToken {
21      function balanceOf(address account, uint256 id) external view returns (uint256);
22  }
23
24  /**
25   * @title ExchangeHandlerFacet
26   *
27   * @notice Handles exchanges associated with offers within the protocol
28   */
29  contract ExchangeHandlerFacet is IBosonExchangeHandler, BuyerBase, DisputeBase {
```

### Recommendation:

We advise the latter two of the three to be split to their dedicated files to aid in the codebase's maintainability.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

Both `interface` declarations have been omitted from the contract thus optimizing its legibility.

# FundsHandlerFacet Code Style Findings

## FHF-01C: Inefficient `mapping` Lookups

| Type | Severity | Location |
| --- | --- | --- |
| **Gas Optimization** | Informational | FundsHandlerFacet.sol:L174, L225 |

### 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/FundsHandlerFacet.sol

SOL

```sol
223 for (uint256 i = 0; i < len; i++) {
224     // get available funds from storage
225     uint256 availableFunds = protocolLookups().availableFunds[_entityId][tokenList[i]]
226     FundsLib.transferFundsFromProtocol(_entityId, tokenList[i], _destinationAddress, a
227 }
```

### 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 (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The referenced `mapping` lookups have been optimized to the greatest extent possible thus greatly reducing the gas cost of the codebase.

# FHF-02C: Repetitive Invocations of Getter Functions

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | FundsHandlerFacet.sol:L94, L97, L100, L106, L109, L154, L174, L215, L225 |

### Description:

The linked statements repetitively invoke the same getter functions whilst their return values remain unchanged.

### Example:

contracts/protocol/facets/FundsHandlerFacet.sol

SOL

```sol
function withdrawFunds(
    uint256 _entityId,
    address[] calldata _tokenList,
    uint256[] calldata _tokenAmounts
) external override fundsNotPaused nonReentrant {
    // address that will receive the funds
    address payable destinationAddress;

    // first check if the caller is a buyer
    (bool exists, uint256 callerId) = getBuyerIdByWallet(msgSender());
    if (exists && callerId == _entityId) {
        // caller is a buyer
        destinationAddress = payable(msgSender());
    } else {
        // check if the caller is a clerk
        (exists, callerId) = getSellerIdByClerk(msgSender());
        if (exists && callerId == _entityId) {
            // caller is a clerk. In this case funds are transferred to the treasury a
            (, Seller storage seller, ) = fetchSeller(callerId);
            destinationAddress = seller.treasury;
        } else {
            (exists, callerId) = getAgentIdByWallet(msgSender());
            if (exists && callerId == _entityId) {
                // caller is an agent
                destinationAddress = payable(msgSender());
```

```
110             } else {
111                 // in this branch, caller is neither buyer, clerk or agent or does not
112                 revert(NOT_AUTHORIZED);
113             }
114         }
115     }
116
117     withdrawFundsInternal(destinationAddress, _entityId, _tokenList, _tokenAmounts);
118 }
```

### Recommendation:

We advise the getter functions to be invoked once and their results to be stored to local variables that are consequently utilized optimizing the gas cost of the statements significantly.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

All repetitive getter function invocations have been reduced to a single getter invocation per repetitive instance thus optimizing the codebase significantly.

# FHF-03C: Suboptimal Struct Declaration Style

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | FundsHandlerFacet.sol:L177 |

### Description:

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

### Example:

contracts/protocol/facets/FundsHandlerFacet.sol

SOL

```
177 availableFunds[i] = Funds(tokenAddress, tokenName, availableAmount);
```

### Recommendation:

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

### Alleviation (44009967e4):

The `Funds` struct creation has instead been adjusted to three separate assignment statements which we consider sub-optimal. We advised the usage of the key-value declaration style (i.e. `Funds(a, b, c)` would become `Funds({tokenAddress:a,tokenName:b,availableAmount:c})`) to optimize the code's legibility.

### Alleviation (6dae5d2602):

The Boson Protocol team stated that they wish to retain the newly introduced per-key assignment format to ensure consistency across the codebase and to also permit instantiation of structs with members that are otherwise impossible to copy by memory (i.e. in-memory arrays). Given that the usage of the per-key style is a valid case for types that cannot be assigned in a direct `struct` instantiation statement, we consider this exhibit adequately dealt with as the ambiguity portion of the exhibit has been sufficiently dealt with.

# FundsLib Code Style Findings

## FLB-01C: Ineffectual Usage of Safe Arithmetics

| Type | Severity | Location |
| --- | --- | --- |
| **Language Specific** | Informational | FundsLib.sol:L328 |

### 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/libs/FundsLib.sol

SOL

```
326 // make sure that seller has enough funds in the pool and reduce the available funds
327 require(availableFunds >= _amount, INSUFFICIENT_AVAILABLE_FUNDS);
328 pl.availableFunds[_entityId][_tokenAddress] = availableFunds - _amount;
```

### 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 (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

An `unchecked` code block has been introduced surrounding the linked arithmetic statement thus optimizing its execution cost safely.

# FLB-02C: Inefficient `mapping` Lookups

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | FundsLib.sol:L295, L296, L298, L302, L324, L328, L332, L334, L338, L340, L342, L345, L347 |

### Description:

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

### Example:

contracts/protocol/libs/FundsLib.sol

SOL

```sol
323 // get available funds from storage
324 uint256 availableFunds = pl.availableFunds[_entityId][_tokenAddress];
325
326 // make sure that seller has enough funds in the pool and reduce the available funds
327 require(availableFunds >= _amount, INSUFFICIENT_AVAILABLE_FUNDS);
328 pl.availableFunds[_entityId][_tokenAddress] = availableFunds - _amount;
329
330 // if availableFunds are totally emptied, the token address is removed from the seller
331 if (availableFunds == _amount) {
332     uint256 lastTokenIndex = pl.tokenList[_entityId].length - 1;
333     //Get the index in the tokenList array, which is 1 less than the tokenIndexByAccou
334     uint256 index = pl.tokenIndexByAccount[_entityId][_tokenAddress] - 1;
335     if (index != lastTokenIndex) {
336         // if index == len - 1 then only pop and delete are needed
337         // Need to fill gap caused by delete if more than one element in storage array
338         address tokenToMove = pl.tokenList[_entityId][lastTokenIndex];
339         // Copy the last token in the array to this index to fill the gap
340         pl.tokenList[_entityId][index] = tokenToMove;
341         // Reset index mapping. Should be index in tokenList array + 1
342         pl.tokenIndexByAccount[_entityId][tokenToMove] = index + 1;
343     }
344     // Delete last token address in the array, which was just moved to fill the gap
345     pl.tokenList[_entityId].pop();
346     //Delete from index mapping
347     delete pl.tokenIndexByAccount[_entityId][_tokenAddress];
```

```
348 }
```

### 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 (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The referenced `mapping` lookups have been optimized to the greatest extent possible thus greatly reducing the gas cost of the codebase.

# FLB-03C: Inexplicable Contract Specifier

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | FundsLib.sol:L261 |

### Description:

The `decreaseAvailableFunds` function is invoked using the contract specifier `FundsLib` which is the contract itself.

### Example:

contracts/protocol/libs/FundsLib.sol

SOL

```
261 FundsLib.decreaseAvailableFunds(_entityId, _tokenAddress, _amount);
```

### Recommendation:

We advise the contract specifier to be omitted as `decreaseAvailableFunds` will refer to the function defined within the contract in this case.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The `FundsLib` specifier has been removed from the referenced function invocation optimizing the code's legibility.

# GroupBase Code Style Findings

## GBE-01C: Repetitive Invocations of Getter Functions

| Type | Severity | Location |
| --- | --- | --- |
| **Gas Optimization** | Informational | GroupBase.sol:L56, L59, L155, L161 |

### Description:

The linked statements repetitively invoke the same getter functions whilst their return values remain unchanged.

### Example:

contracts/protocol/bases/GroupBase.sol

SOL

```sol
55  // add to groupIdByOffer mapping
56  protocolLookups().groupIdByOffer[_group.offerIds[i]] = groupId;
57
58  // Set index mapping. Should be index in offerIds + 1
59  protocolLookups().offerIdIndexByGroup[groupId][_group.offerIds[i]] = i + 1;
```

### Recommendation:

We advise the getter functions to be invoked once and their results to be stored to local variables that are consequently utilized optimizing the gas cost of the statements significantly.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The protocol lookups are now correctly cached to local variable declarations thus optimizing the codebase's gas cost.

# GroupHandlerFacet Code Style Findings

## GHF-01C: Inefficient `mapping` Lookups

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | GroupHandlerFacet.sol:L113, L122, L127 |

### 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/GroupHandlerFacet.sol

SOL

```sol
uint256 len = group.offerIds.length;
//Get the index in the offerIds array, which is 1 less than the offerIdIndexByGroup in
uint256 index = protocolLookups().offerIdIndexByGroup[groupId][offerId] - 1;

if (index != len - 1) {
    // if index == len - 1 then only pop and delete are needed
    // Need to fill gap caused by delete if more than one element in storage array
    uint256 offerIdToMove = group.offerIds[len - 1];
    // Copy the last token in the array to this index to fill the gap
    group.offerIds[index] = offerIdToMove;
    //Reset index mapping. Should be index in offerIds array + 1
    protocolLookups().offerIdIndexByGroup[groupId][offerIdToMove] = index + 1;
}
// Delete last offer id in the array, which was just moved to fill the gap
group.offerIds.pop();
// Delete from index mapping
delete protocolLookups().offerIdIndexByGroup[groupId][offerId];
```

### 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 (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The referenced `mapping` lookups have been optimized to the greatest extent possible thus greatly reducing the gas cost of the codebase.

# GHF-02C: Repetitive Invocations of Getter Function

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | GroupHandlerFacet.sol:L109, L113, L122, L127 |

## Description:

The linked statements repetitively invoke the same getter function whilst its return value remains unchanged.

## Example:

contracts/protocol/facets/GroupHandlerFacet.sol

SOL

```sol
for (uint256 i = 0; i < _offerIds.length; i++) {
    uint256 offerId = _offerIds[i];

    // Offer should belong to the group
    (, uint256 groupId) = getGroupIdByOffer(offerId);
    require(_groupId == groupId, OFFER_NOT_IN_GROUP);

    // remove groupIdByOffer mapping
    delete protocolLookups().groupIdByOffer[offerId];

    uint256 len = group.offerIds.length;
    //Get the index in the offerIds array, which is 1 less than the offerIdIndexByGrou
    uint256 index = protocolLookups().offerIdIndexByGroup[groupId][offerId] - 1;

    if (index != len - 1) {
        // if index == len - 1 then only pop and delete are needed
        // Need to fill gap caused by delete if more than one element in storage array
        uint256 offerIdToMove = group.offerIds[len - 1];
        // Copy the last token in the array to this index to fill the gap
        group.offerIds[index] = offerIdToMove;
        //Reset index mapping. Should be index in offerIds array + 1
        protocolLookups().offerIdIndexByGroup[groupId][offerIdToMove] = index + 1;
    }
    // Delete last offer id in the array, which was just moved to fill the gap
    group.offerIds.pop();
    // Delete from index mapping
    delete protocolLookups().offerIdIndexByGroup[groupId][offerId];
```

```
128 }
```

### Recommendation:

We advise the getter function to be invoked once and its result to be stored to a local variable that is consequently utilized optimizing the gas cost of the statements significantly.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The protocol lookups are now correctly cached to local variable declarations thus optimizing the codebase's gas cost.

# OfferBase Code Style Findings

## OBE-01C: Suboptimal Struct Declaration Style

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | OfferBase.sol:L173-L178 |

### Description:

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

### Example:

contracts/protocol/bases/OfferBase.sol

SOL

```sol
173 disputeResolutionTerms = DisputeResolutionTerms(
174     _disputeResolverId,
175     disputeResolver.escalationResponsePeriod,
176     feeAmount,
177     (feeAmount * pl.buyerEscalationDepositPercentage) / 10000
178 );
```

### Recommendation:

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

### Alleviation (44009967e4):

The `DisputeResolutionTerms` struct creation has instead been adjusted to four separate assignment statements which we consider sub-optimal. We advised the usage of the key-value declaration style (i.e. `Funds(a, b, c)` would become `Funds({tokenAddress:a,tokenName:b,availableAmount:c})`) to optimize the code's legibility.

### Alleviation (6dae5d2602):

The Boson Protocol team stated that they wish to retain the newly introduced per-key assignment format to ensure consistency across the codebase and to also permit instantiation of structs with members that are otherwise impossible to copy by memory (i.e. in-memory arrays). Given that the usage of the per-key style is a valid case for types that cannot be assigned in a direct `struct` instantiation statement, we consider this exhibit adequately dealt with as the ambiguity portion of the exhibit has been sufficiently dealt with.

# OrchestrationHandlerFacet Code Style Findings

## CON-01C: Redundant Modifier Overlap

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | OrchestrationHandlerFacet.sol:L147, L338, L414, L496, L585-L590 |

### Description:

The linked `modifier` applications contain redundant overlap that leads to an increased execution cost as certain `external` functions invoke `public` functions that apply a portion of their `modifier` declarations.

### Example:

contracts/protocol/facets/OrchestrationHandlerFacet.sol

SOL

```
404 function createSellerAndOfferWithCondition(
405     Seller memory _seller,
406     Offer memory _offer,
407     OfferDates calldata _offerDates,
408     OfferDurations calldata _offerDurations,
409     uint256 _disputeResolverId,
410     Condition calldata _condition,
411     AuthToken calldata _authToken,
412     VoucherInitValues calldata _voucherInitValues,
413     uint256 _agentId
414 ) external override sellersNotPaused offersNotPaused groupsNotPaused orchestrationNotP
415     checkAndCreateSeller(_seller, _authToken, _voucherInitValues);
416     createOfferWithCondition(_offer, _offerDates, _offerDurations, _disputeResolverId,
417 }
```

### Recommendation:

We advise the `modifier` overlaps to be detected and omitted from the outer-most call to minimize the execution cost of the contract.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

All `modifier` overlaps have been properly omitted thus optimizing the codebase's gas cost whilst retaining the same security guarantees.

# PausableBase Code Style Findings

## PBE-01C: Bytecode Size Optimization

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | PausableBase.sol:L22, L34, L46, L58, L70, L82, L94, L106, L118, L130, L142, L154, L166 |

### Description:

The linked `require` statements are replicated across multiple `modifier` implementations with the same error message.

### Example:

contracts/protocol/bases/PausableBase.sol

SOL

```sol
14  /**
15   * @dev Modifier that checks the Offers region is not paused
16   *
17   * Reverts if region is paused
18   *
19   * See: {BosonTypes.PausableRegion}
20   */
21  modifier offersNotPaused() {
22      require(!paused(PausableRegion.Offers), REGION_PAUSED);
23      _;
24  }
25
26  /**
27   * @dev Modifier that checks the Twins region is not paused
28   *
29   * Reverts if region is paused
30   *
31   * See: {BosonTypes.PausableRegion}
32   */
33  modifier twinsNotPaused() {
34      require(!paused(PausableRegion.Twins), REGION_PAUSED);
35      _;
36  }
```

### Recommendation:

We advise all `require` statements to instead invoke a `private` function that accepts a `bool` argument and reverts with the `REGION_PAUSED` error message thus significantly reducing the generated bytecode size of all contracts that use the modifiers.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The `paused` function has instead been renamed to `revertIfPaused` and adequately optimizes each `modifier`'s bytecode footprint as it now contains the error message and `require` call instead of yielding the pause status to its caller.

# PBE-02C: Potential Operator Optimization

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | PausableBase.sol:L177 |

### Description:

The linked statement is meant to calculate the bit that should be validated as present in the `pauseScenario` variable, however, the bit is calculated using a power operation (`**`) which is expensive.

### Example:

contracts/protocol/bases/PausableBase.sol

SOL

```sol
170 /**
171  * @dev Check if a region of the protocol is paused.
172  *
173  * @param _region the region to check pause status for
174  */
175 function paused(PausableRegion _region) internal view returns (bool) {
176     // Region enum value must be used as the exponent in a power of 2
177     uint256 powerOfTwo = 2**uint256(_region);
178     return (ProtocolLib.protocolStatus().pauseScenario & powerOfTwo) == powerOfTwo;
179 }
```

### Recommendation:

We advise a bitwise shift to be utilized instead, adjusting the assignment to `1 << uint256(_region)` achieving the same result in cheaper operations.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

A bitwise shift operation is now correctly utilized to assess the power-of-two operation optimizing the codebase's gas cost.

# ProtocolBase Code Style Findings

## PBS-01C: Repetitive Invocations of Getter Functions

| Type | Severity | Location |
| --- | --- | --- |
| **Gas Optimization** | Informational | ProtocolBase.sol:L297, L300, L339, L342, L488, L491 |

### Description:

The linked statements repetitively invoke the same getter functions whilst their return values remain unchanged.

### Example:

contracts/protocol/bases/ProtocolBase.sol

SOL

```
487 // Get the dispute's slot
488 dispute = protocolEntities().disputes[_exchangeId];
489
490 // Get the disputeDates's slot
491 disputeDates = protocolEntities().disputeDates[_exchangeId];
```

### Recommendation:

We advise the getter functions to be invoked once and their results to be stored to local variables that are consequently utilized optimizing the gas cost of the statements significantly.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The protocol entities are now correctly cached to local variable declarations thus optimizing the codebase's gas cost.

# ProtocolLib Code Style Findings

## PLB-01C: Generic Typographic Mistake

| Type | Severity | Location |
| --- | --- | --- |
| **Code Style** | Informational | ProtocolLib.sol:L57 |

### Description:

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

### Example:

contracts/protocol/libs/ProtocolLib.sol

SOL

```
57  // limit the sum of (Protocol Fee percentage + Agent Fee perpercentagecent) of an offe
```

### Recommendation:

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

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The typographic mistake has been appropriately corrected.

## PLB-02C: Inefficient Data Types

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | ProtocolLib.sol:L66, L156 |

### Description:

The `uint16` data type utilized in the linked instances is redundant as it does not take advantage of Solidity's tight-packing mechanism and represents a sub-256 bit data type that is more expensive to operate on.

### Example:

contracts/protocol/libs/ProtocolLib.sol

SOL

```sol
63  // Protocol fees storage
64  struct ProtocolFees {
65      // Percentage that will be taken as a fee from the net of a Boson Protocol exchang
66      uint16 percentage; // 1.75% = 175, 100% = 10000
67      // Flat fee taken for exchanges in $BOSON
68      uint256 flatBoson;
69  }
```

### Recommendation:

We advise a full `uint256` variable and storage slot to be utilized by the referenced data entries as the EVM is natively more performant when operating on 256-bit data types than when operating on sub-256 data types as on the latter it has to perform low-level padding and unpadding operations on the operated members.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The first data type has been up-scaled to a `uint256` properly whilst the latter data type is no longer present in the codebase thus addressing this exhibit in full.

# ReentrancyGuardBase Code Style Findings

## RGB-01C: Outdated Documentation of Reentrancy Guard

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | ReentrancyGuardBase.sol:L38, L46-L47 |

### Description:

The linked documentation lines of the `ReentrancyGuardBase` contract are taken from the original `bool`-based implementation by OpenZeppelin, however, the implementation used by the Boson Protocol is `uint256` based as the linked **EIP-2200** regarding gas refunds is no longer valid.

### Example:

contracts/protocol/bases/ReentrancyGuardBase.sol

SOL

```sol
36  modifier nonReentrant() {
37      ProtocolLib.ProtocolStatus storage ps = ProtocolLib.protocolStatus();
38      // On the first call to nonReentrant, _notEntered will be true
39      require(ps.reentrancyStatus != ENTERED, REENTRANCY_GUARD);
40
41      // Any calls to nonReentrant after this point will fail
42      ps.reentrancyStatus = ENTERED;
43
44      _;
45
46      // By storing the original value once again, a refund is triggered (see
47      // https://eips.ethereum.org/EIPS/eip-2200)
48      ps.reentrancyStatus = NOT_ENTERED;
49  }
```

### Recommendation:

We advise the documentation lines of the contract to be corrected reflecting the new functionality it represents.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

**Alleviation (44009967e4f6809294fd841e9e0f5dd2bb31b10b):**

The documentation of the contract has been properly updated to reflect its logic as advised.

# SellerBase Code Style Findings

## SBE-01C: Multiple Top-Level Type Declarations

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | SellerBase.sol:L15, L170 |

### Description:

The `SellerBase` file contains two top-level declarations (`SellerBase` and `IInitializableClone`).

### Example:

contracts/protocol/bases/SellerBase.sol

SOL

```
168 }
169
170 interface IInitializableClone {
```

### Recommendation:

We advise the latter of the two to be split to its dedicated file to aid in the codebase's maintainability.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The `IInitializableClone` interface declaration has been renamed and relocated to its dedicated file that is imported to the codebase thus alleviating this exhibit.

## SBE-02C: Repetitive Invocations of Getter Functions

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | SellerBase.sol:L49-L54, L62, L68-L70, L82, L124, L128, L132-L133 |

### Description:

The linked statements repetitively invoke the same getter functions whilst their return values remain unchanged.

### Example:

contracts/protocol/bases/SellerBase.sol

SOL

```
48  require(
49      protocolLookups().sellerIdByOperator[_seller.operator] == 0 &&
50          protocolLookups().sellerIdByOperator[_seller.clerk] == 0 &&
51          protocolLookups().sellerIdByAdmin[_seller.operator] == 0 &&
52          protocolLookups().sellerIdByAdmin[_seller.clerk] == 0 &&
53          protocolLookups().sellerIdByClerk[_seller.operator] == 0 &&
54          protocolLookups().sellerIdByClerk[_seller.clerk] == 0,
55      SELLER_ADDRESS_MUST_BE_UNIQUE
56  );
```

### Recommendation:

We advise the getter functions to be invoked once and their results to be stored to local variables that are consequently utilized optimizing the gas cost of the statements significantly.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The protocol lookups are now correctly cached to local variable declarations thus optimizing the codebase's gas cost.

# SellerHandlerFacet Code Style Findings

## SHF-01C: Inefficient `mapping` Lookups

| Type | Severity | Location |
| --- | --- | --- |
| **Gas Optimization** | Informational | SellerHandlerFacet.sol:L95-L106, L114-L115, L122-L127 |

### 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

```
113 require(
114     protocolLookups().sellerIdByAuthToken[_authToken.tokenType][_authToken.tokenId] ==
115         protocolLookups().sellerIdByAuthToken[_authToken.tokenType][_authToken.tokenId
116     AUTH_TOKEN_MUST_BE_UNIQUE
117 );
```

### 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 (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The referenced `mapping` lookups have been optimized to the greatest extent possible thus greatly reducing the gas cost of the codebase.

## SHF-02C: Repetitive Invocations of Getter Function

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | SellerHandlerFacet.sol:L95-L106, L114-L115, L122-L127, L133-L136, L148 |

### Description:

The linked statements repetitively invoke the same getter function whilst its return value remains unchanged.

### Example:

contracts/protocol/facets/SellerHandlerFacet.sol

SOL

```
95  (protocolLookups().sellerIdByOperator[_seller.operator] == 0 ||
96      protocolLookups().sellerIdByOperator[_seller.operator] == _seller.id) &&
97      (protocolLookups().sellerIdByOperator[_seller.clerk] == 0 ||
98          protocolLookups().sellerIdByOperator[_seller.clerk] == _seller.id) &&
99      (protocolLookups().sellerIdByAdmin[_seller.operator] == 0 ||
100         protocolLookups().sellerIdByAdmin[_seller.operator] == _seller.id) &&
101     (protocolLookups().sellerIdByAdmin[_seller.clerk] == 0 ||
102         protocolLookups().sellerIdByAdmin[_seller.clerk] == _seller.id) &&
103     (protocolLookups().sellerIdByClerk[_seller.operator] == 0 ||
104         protocolLookups().sellerIdByClerk[_seller.operator] == _seller.id) &&
105     (protocolLookups().sellerIdByClerk[_seller.clerk] == 0 ||
106         protocolLookups().sellerIdByClerk[_seller.clerk] == _seller.id),
```

### Recommendation:

We advise the getter function to be invoked once and its result to be stored to a local variable that is consequently utilized optimizing the gas cost of the statements significantly.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The protocol lookups are now correctly cached to local variable declarations thus optimizing the codebase's gas cost.

# TwinBase Code Style Findings

## TBE-01C: Hard-Coded Interface Types

| Type | Severity | Location |
| --- | --- | --- |
| **Code Style** | Informational | TwinBase.sol:L49, L100 |

### Description:

The `IERC721` & `IERC1155` interface IDs that are hard-coded in the referenced statements are ill-advised.

### Example:

contracts/protocol/bases/TwinBase.sol

SOL

```
48  // Check if the token supports IERC721 interface
49  require(contractSupportsInterface(_twin.tokenAddress, 0x80ac58cd), INVALID_TOKEN_ADDRE
```

### Recommendation:

We advise the `IERC721` & `IERC1155` interfaces to be imported in the codebase and the `interfaceId` specifier of each to be utilized for specifying the interface the `tokenAddress` contract is meant to support.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The `interfaceId` specifier of the `IERC721` and `IERC1155` interfaces respectively are now correctly utilized in the referenced statements optimizing the legibility of the codebase.

# TBE-02C: Inefficient Loop Limit Evaluations

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | TwinBase.sol:L70, L84 |

### Description:

The linked `for` loops evaluate their limit inefficiently on each iteration.

### Example:

contracts/protocol/bases/TwinBase.sol

SOL

```sol
70  for (uint256 i = 0; i < twinIds.length; i++) {
```

### Recommendation:

We advise the statements within the `for` loop limits to be relocated outside to a local variable declaration that is consequently utilized for the evaluations to significantly reduce the codebase's gas cost. We should note the same optimization is applicable for storage reads present in those limits as they are newly read on each iteration (i.e. `length` members of arrays in storage).

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The `length` member in both `for` loop instances is now cached to a local variable that is instead evaluated optimizing the gas cost of the referenced loops.

# TBE-03C: Inefficient `mapping` Lookups

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | TwinBase.sol:L66-L68, L81, L92, L94 |

### Description:

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

### Example:

contracts/protocol/bases/TwinBase.sol

SOL

```sol
// Get all seller twin ids that belong to the same token address of the new twin to va
uint256[] storage twinIds = protocolLookups().twinIdsByTokenAddressAndBySeller[sellerI
    _twin.tokenAddress
];

for (uint256 i = 0; i < twinIds.length; i++) {
    // Get storage location for looped twin
    (, Twin storage currentTwin) = fetchTwin(twinIds[i]);

    // The protocol cannot allow two twins with unlimited supply and with the same tok
    if (currentTwin.supplyAvailable == type(uint256).max || _twin.supplyAvailable == t
        require(currentTwin.tokenAddress != _twin.tokenAddress, INVALID_TWIN_TOKEN_RAN
    }
}

// Get all ranges of twins that belong to the seller and to the same token address of
TokenRange[] storage twinRanges = protocolLookups().twinRangesBySeller[sellerId][_twin

// Checks if token range isn't being used in any other twin of seller
for (uint256 i = 0; i < twinRanges.length; i++) {
    // A valid range has:
    // - the first id of range greater than the last token id (tokenId + initialSupply
    // - the last id of range lower than the looped twin tokenId (beginning of range)
    require(tokenId > twinRanges[i].end || lastTokenId < twinRanges[i].start, INVALID_
}

```

```
91  // Add range to twinRangesBySeller mapping
92  protocolLookups().twinRangesBySeller[sellerId][_twin.tokenAddress].push(TokenRange(tok
93  // Add twin id to twinIdsByTokenAddressAndBySeller mapping
94  protocolLookups().twinIdsByTokenAddressAndBySeller[sellerId][_twin.tokenAddress].push(
```

### 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 (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The referenced `mapping` lookups have been optimized to the greatest extent possible thus greatly reducing the gas cost of the codebase.

# TBE-04C: Repetitive Invocations of Getter Function

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | TwinBase.sol:L66, L81, L92, L94 |

### Description:

The linked statements repetitively invoke the same getter function whilst its return value remains unchanged.

### Example:

contracts/protocol/bases/TwinBase.sol

SOL

```sol
// Get all seller twin ids that belong to the same token address of the new twin to va
uint256[] storage twinIds = protocolLookups().twinIdsByTokenAddressAndBySeller[sellerI
    _twin.tokenAddress
];

for (uint256 i = 0; i < twinIds.length; i++) {
    // Get storage location for looped twin
    (, Twin storage currentTwin) = fetchTwin(twinIds[i]);

    // The protocol cannot allow two twins with unlimited supply and with the same tok
    if (currentTwin.supplyAvailable == type(uint256).max || _twin.supplyAvailable == t
        require(currentTwin.tokenAddress != _twin.tokenAddress, INVALID_TWIN_TOKEN_RAN
    }
}

// Get all ranges of twins that belong to the seller and to the same token address of
TokenRange[] storage twinRanges = protocolLookups().twinRangesBySeller[sellerId][_twin
```

### Recommendation:

We advise the getter function to be invoked once and its result to be stored to a local variable that is consequently utilized optimizing the gas cost of the statements significantly.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The protocol lookups are now correctly cached to local variable declarations thus optimizing the codebase's gas cost.

# TBE-05C: Suboptimal Struct Declaration Type

| Type | Severity | Location |
| --- | --- | --- |
| **Gas Optimization** | Informational | TwinBase.sol:L92 |

### Description:

The linked declaration type of a struct is set as `memory` when not all its variables are utilized within the function.

### Example:

contracts/protocol/bases/TwinBase.sol

SOL

```
92  protocolLookups().twinRangesBySeller[sellerId][_twin.tokenAddress].push(TokenRange(tok
```

### Recommendation:

We advise it to be set as `storage` optimizing its read access gas cost and all variables that are read more than once to be cached to local variables.

### Alleviation (44009967e4):

The `TokenRange` struct creation has instead been adjusted to two separate assignment statements which we consider sub-optimal. We advised the usage of the key-value declaration style (i.e. `Funds(a, b, c)` would become `Funds({tokenAddress:a,tokenName:b,availableAmount:c})`) to optimize the code's legibility.

### Alleviation (6dae5d2602):

The Boson Protocol team stated that they wish to retain the newly introduced per-key assignment format to ensure consistency across the codebase and to also permit instantiation of structs with members that are otherwise impossible to copy by memory (i.e. in-memory arrays). Given that the usage of the per-key style is a valid case for types that cannot be assigned in a direct `struct` instantiation statement, we consider this exhibit adequately dealt with as the ambiguity portion of the exhibit has been sufficiently dealt with.

# TwinHandlerFacet Code Style Findings

## THF-01C: Repetitive Invocations of Getter Function

| Type | Severity | Location |
| --- | --- | --- |
| **Gas Optimization** | Informational | TwinHandlerFacet.sol:L80, L81 |

### Description:

The linked statements repetitively invoke the same getter function whilst its return value remains unchanged.

### Example:

contracts/protocol/facets/TwinHandlerFacet.sol

SOL

```
80  TokenRange[] storage twinRanges = protocolLookups().twinRangesBySeller[sellerId][twin.
81  uint256[] storage twinIdsByTokenAddressAndBySeller = protocolLookups().twinIdsByTokenA
```

### Recommendation:

We advise the getter function to be invoked once and its result to be stored to a local variable that is consequently utilized optimizing the gas cost of the statements significantly.

### Alleviation (44009967e4f68092941d841e9e0f5dd2bb31bf0b):

The protocol lookups are now correctly cached to local variable declarations thus optimizing the codebase's 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.

## External Call Validation

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

## Input Sanitization

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

## Indeterminate Code

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

## Language Specific

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

## Code Style

An official Solidity style guide exists that is constantly under development and is adjusted on each new Solidity release, designating how the overall look and feel of a codebase should be. In these types of findings, we identify whether a project conforms to a particular naming convention and whether that convention is consistent within the codebase and legible. In case of inconsistencies, we point them out under this category. Additionally, variable shadowing falls under this category as well which is identified when a

local-level variable contains the same name as a contract-level variable that is present in the inheritance chain of the local execution level's context.

## Gas Optimization

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

## Standard Conformity

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

## Mathematical Operations

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

## Logical Fault

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

## Centralization Concern

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

## Reentrant Call

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

# Disclaimer

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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