OMNISCIA

September 26, 2023

SMART CONTRACT AUDIT REPORT

Boson Protocol
Version 2.3

omniscia.io

info@omniscia.io

Online report: boson-protocol-version-2.3

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

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

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

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

omniscia.io


info@omniscia.io

Version 2.3 Security Audit

Audit Report Revisions

Commit Hash	Date	Audit Report Hash
02a4d2ff04	July 29th 2023	e2f3430d82
2b9f60b6c3	September 4th 2023	0c75f30525
584e7d054c	September 26th 2023	f48b701844

Audit Overview

We were tasked with performing an audit of the Boson Protocol codebase and in particular the changes introduced between version `2.2.0` and `2.3.0`, including the interim `2.2.1` version released.

Over the course of the audit, we identified multiple potential vulnerabilities of significant severity that arise from the introduction of new features without proper consideration of side-effects that they introduce in the old system.

Namely, the major vulnerabilities we identified:

Additionally, we evaluated the security of the full changelog between the `2.2.0` and `2.3.0` versions, including ensuring that the removal of protocol-wide limitations is secure, asserting that the new condition style is properly assimilated in the codebase, assessing the impact of `create2`-based deployment mechanisms for `BosonVoucher` instances, and more.

As a result of our evaluation, we were able to pinpoint vulnerabilities that concern these new features as well as potential ways the protocol can enhance itself further in terms of functionality as well as backwards-compatibility, a trait we identified is not present in certain sensitive functions of the protocol.

Given that this is a delta audit, certain findings that have been included in older audit rounds have not been replicated in this audit report for the sake of brevity; all findings included concern newly introduced code as well as affected code of newly introduced functionality.

We advise the Boson Protocol team to closely evaluate all minor-and-above findings identified in the report and promptly remediate them as well as consider all optimizational exhibits identified in the report.

Post-Audit Conclusion

The Boson Protocol team iterated through all findings within the report and provided us with a revised commit hash to evaluate all exhibits on.

We evaluated all alleviations performed by Boson Protocol and have identified that certain exhibits have not been adequately dealt with. Specifically, some were partially alleviated while others have had additional information introduced to them and we advise them to be re-visited: `SBE-02M`, `EHF-01M`, `EHF-02C`, `BVR-05M`, `BVR-04M`

Post-Audit Conclusion (584e7d054c)

The Boson Protocol assessed the remaining exhibits highlighted above and proceeded to correctly apply the remediation of exhibit `EHF-02C` while providing supplemental information in the threads of the rest.

We revisited these exhibits and re-evaluated them based on this information, either marking them acknowledged, nullified, or properly alleviated.

No outstanding exhibits remain in the report and all outputs of the audit have been properly consumed by the Boson Protocol team.

Contracts Assessed

Files in Scope	Repository	Commit(s)
Address.sol (ASS)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d054c
AccessControl.sol (ACL)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d054c
AccessController.sol (ACR)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d054c
AgentHandlerFacet.sol (AHF)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d054c
AccountHandlerFacet.sol (AHT)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d054c
BuyerBase.sol (BBE)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d054c
BosonTypes.sol (BTS)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d054c
BundleBase.sol (BBS)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d054c
BosonVoucher.sol (BVR)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d054c
BosonConstants.sol (BCS)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d054c
BeaconClientLib.sol (BCL)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d054c
BeaconClientBase.sol (BCB)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d054c
BeaconClientProxy.sol (BCP)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d054c
BosonClientBeacon.sol (BCN)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d054c
BuyerHandlerFacet.sol (BHF)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d054c
BundleHandlerFacet.sol (BHT)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d054c

ClientLib.sol (CLB)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d05
ClientBase.sol (CBE)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d05
ClientProxy.sol (CPY)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d05
ConfigHandlerFacet.sol (CHF)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d05
ClientExternalAddressesBase.sol (CEA)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d05
DiamondLib.sol (DLB)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d05
DisputeBase.sol (DBE)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d05
DiamondCutFacet.sol (DCF)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d05
DiamondLoupeFacet.sol (DLF)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d05
DisputeHandlerFacet.sol (DHF)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d05
DisputeResolverHandlerFacet.sol (DRH)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d05
EIP712Lib.sol (EIP)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d05
ERC165Facet.sol (ERC)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d05
ExchangeHandlerFacet.sol (EHF)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d05
FundsLib.sol (FLB)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d05
FundsHandlerFacet.sol (FHF)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d05
GroupBase.sol (GBE)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d05
GroupHandlerFacet.sol (GHF)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d05
JewelerLib.sol (JLB)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d05
Math.sol (MHT)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7d05

MetaTransactionsHandlerFacet.sol (MTH)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7c
OfferBase.sol (OBE)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7c
OfferHandlerFacet.sol (OHF)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7c
OrchestrationHandlerFacet1.sol (OH1)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7c
OrchestrationHandlerFacet2.sol (OH2)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7c
Proxy.sol (PYX)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7c
ProtocolLib.sol (PLB)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7c
PausableBase.sol (PBE)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7c
ProtocolBase.sol (PBS)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7c
ProtocolDiamond.sol (PDD)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7c
PauseHandlerFacet.sol (PHF)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7c
ProtocolInitializationHandlerFacet.sol (PIH)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7c
ReentrancyGuardBase.sol (RGB)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7c
Strings.sol (SSG)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7c
SafeERC20.sol (SER)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7c
SellerBase.sol (SBE)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7c
SellerHandlerFacet.sol (SHF)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7c
TwinBase.sol (TBE)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7c
TwinHandlerFacet.sol (THF)	boson-protocol-contracts	02a4d2ff04, 2b9f60b6c3, 584e7c

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	3	3	0	0
Informational	14	12	0	2
Minor	6	5	0	1
Medium	5	4	0	1
Major	4	4	0	0

During the audit, we filtered and validated a total of **2 findings utilizing static analysis** tools as well as identified a total of **30 findings during the manual review** of the codebase. We strongly recommend that any minor severity or higher findings are dealt with promptly prior to the project's launch as they can introduce potential misbehaviours of the system as well as exploits.

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

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

The `pragma` versions have been locked to `0.8.18` (`=0.8.18`), 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 **113 potential issues** within the codebase of which **111 were ruled out to be false positives** or negligible findings.

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

ID	Severity	Addressed	Title
FHF-01S	Informational	Acknowledged	Illegible Numeric Value Representation
OBE-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 core implementation and in particular the delta between versions `2.2.0` and `2.3.0`.

As the project at hand comprises a delta audit of newly introduced functionality to the Boson Protocol system, intricate care was put into ensuring that the **flow of funds & assets within the system conforms to the specifications and restrictions** laid forth within the protocol's specification.

We validated that **all state transitions of the system occur within sane criteria** and that all rudimentary formulas within the system execute as expected. We **pinpointed multiple vulnerabilities of significant concern** within the system which could have had **severe ramifications** to its overall operation; we urge the Boson Protocol team to evaluate and rectify them as soon as possible.

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, containing extensive in-line documentation throughout the project, properly maintained changelogs, as well as commits and pull-requests that are verbose in nature.

A total of **30 findings** were identified over the course of the manual review of which **19 findings** concerned the behaviour and security of the system. The non-security related findings, such as optimizations, are included in the separate **Code Style** chapter.

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

ID	Severity	Addressed	Title
BVR-01M	Unknown	Nullified	Inexistent Restriction of Approval for Owner
BVR-02M	Unknown	Nullified	Potentially Malformed Contract Storage Space
BVR-03M	Minor	Acknowledged	Inexistent Access Control of Protocol Withdrawals
BVR-04M	Medium	Nullified	Inexistent Legacy Compatibility of Boson Voucher Premints
BVR-05M	Medium	Acknowledged	Inexistent Transfer of Preminted Voucher Ranges
BVR-06M	Major	Yes	Insufficient Protection of Contract Assets
BVR-07M	Major	Nullified	Storage Conflict of Beacon Implementation
BBS-01M	Medium	Nullified	Removal of Bundle Limitations
CHF-01M	Minor	Yes	Inexistent Validation of Proper Resolution Period Configuration
EHF-01M	Unknown	Nullified	Breaking Change of Single-Point-of-Entry

EHF-02M	Informational	Yes	Restrictive Migration Mechanism
EHF-03M	Major	Yes	Bypass of Token Specific Conditions
GBE-01M	Medium	Nullified	Insufficient Validation of Conditions
PHF-01M	Minor	Yes	Incorrect Revert Condition Specification
PIH-01M	Minor	Yes	Inexistent Support of Manual Seller Configuration
PIH-02M	Minor	Yes	Insufficient Sanitization of Minimum Resolution Period
SBE-01M	Minor	Yes	Restrictive Deployment of Voucher Clone
SBE-02M	Medium	Yes	Incorrect Association of Seller ID
SHF-01M	Major	Yes	Incorrect Iterator Usage

Code Style

During the manual portion of the audit, we identified **11 optimizations** that can be applied to the codebase that will decrease the operational cost associated with the execution of a particular function and generally ensure that the project complies with the latest best practices and standards in Solidity.

Additionally, this section of the audit contains any opinionated adjustments we believe the code should make to make it more legible as well as truer to its purpose.

These optimizations are enumerated below:

ID	Severity	Addressed	Title
BVR-01C	Informational	Yes	Potential Legibility Optimization
BVR-02C	Informational	Yes	Redundant Parenthesis Statements
EHF-01C	Informational	Yes	Inefficient `mapping` Lookups
EHF-02C	Informational	Yes	Redundant Parenthesis Statement
FHF-01C	Informational	Yes	Loop Iterator Optimizations
OBE-01C	Informational	Yes	Illegible Representation of Limitation
PHF-01C	Informational	Yes	Loop Iterator Optimizations
PHF-02C	Informational	Yes	Redundant Parenthesis Statement
PIH-01C	Informational	Nullified	Inexistent Support for Sequential Upgrade
PIH-02C	Informational	Nullified	Loop Iterator Optimization

SHF-01C	Informational	Yes	Loop Iterator Optimization

FundsHandlerFacet Static Analysis Findings

FHF-01S: Illegible Numeric Value Representation

Type	Severity	Location
Code Style	Informational	FundsHandlerFacet.sol:L233

Description:

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

Example:

contracts/protocol/facets/FundsHandlerFacet.sol

SOL

```sol
233 try IERC20Metadata(tokenAddress).name{ gas: 40000 }() returns (string memory name) {
```

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 (2b9f60b6c3323fd234b570089ceff924cdb5851c):

The Boson Protocol has stated that they do not follow this naming convention elsewhere and as such wish to retain the current one in place.

OfferBase Static Analysis Findings

OBE-01S: Illegible Numeric Value Representation

Type	Severity	Location
Code Style	Informational	OfferBase.sol:L194

Description:

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

Example:

contracts/protocol/bases/OfferBase.sol

SOL

```
194 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 (2b9f60b6c3323fd234b570089ceff924cdb5851c):

The Boson Protocol has stated that they do not follow this naming convention elsewhere and as such wish to retain the current one in place.

BosonVoucher Manual Review Findings

BVR-01M: Inexistent Restriction of Approval for Owner

Type	Severity	Location
Logical Fault	Unknown	BosonVoucher.sol:L558, L561

Description:

The Boson Protocol **PR#571** and specifically **this thread** indicates that the `_operator` being approved should not be the `OwnableUpgradeable::owner`, however, the function reverts solely if the `_operator` is the `address(this)` value.

Example:

contracts/protocol/clients/voucher/BosonVoucher.sol

SOL

```sol
558 function setApprovalForAllToContract(address _operator, bool _approved) external
onlyOwner {
559     require(_operator != address(0), INVALID_ADDRESS);
560
561     _setApprovalForAll(address(this), _operator, _approved);
562 }
```

Recommendation:

We advise the Boson Protocol to revisit this particular change and ensure that the `owner` of the referenced comment refers to the "owner" of the **EIP-721** vouchers rather than the `OwnableUpgradeable::owner`.

Alleviation (2b9f60b6c3323fd234b570089ceff924cdb5851c):

The Boson Protocol team clarified that the owner in question is the voucher's owner and as such, the code behaves as expected rendering this exhibit nullified.

BVR-02M: Potentially Malformed Contract Storage Space

Type	Severity	Location
Language Specific	Unknown	BosonVoucher.sol:L833

Description:

The `ERC2771ContextUpgradeable` contract implementation used to reserve one storage slot in its original `v4.0.0` OpenZeppelin implementation and up to `v4.4.0`, however, it was updated to make use of `immutable` variables in `v4.5.0` and up thus causing it to no longer require a storage space.

While the dependency itself maintains a proper storage structure due to the usage of a `__gap` value that was updated, the `BosonVoucherBase` does not have such a gap and adjusts its storage space between implementations.

Impact:

Given that this vulnerability would solely arise if `ERC2771ContextUpgradeable` was updated by OpenZeppelin to make use of storage slots and / or if the Boson Protocol inherits implementations that make use of storage space for the `BosonVoucher`, its severity is indeterminate as it may never manifest in practice.

Example:

contracts/protocol/clients/voucher/BosonVoucher.sol

SOL

```
783 /*
784  * Returns storage pointer to location of private variables
785  * 0x99 is location of _owners
786  * 0x9a is location of _balances
787  *
788  * Since ERC721UpgradeableStorage slot is 0x99
789  * _owners slot is ERC721UpgradeableStorage + 0
790  * _balances slot is ERC721UpgradeableStorage + 1
791  */
```

Recommendation:

We advise the `__gap` value that was removed from `BosonVoucher` to be relocated to the `BosonVoucherBase` implementation, ensuring the storage space of `BosonVoucher` is unaffected between upgrades of the `BosonVoucherBase` dependency.

Alleviation (2b9f60b6c3323fd234b570089ceff924cdb5851c):

The `BosonVoucherBase` already possessed a `__gap` variable, rendering this exhibit nullified.

BVR-03M: Inexistent Access Control of Protocol Withdrawals

Type	Severity	Location
Logical Fault	Minor	BosonVoucher.sol:L766

Description:

The `BosonVoucherBase::withdrawToProtocol` function does not apply any access control to its caller, permitting anyone to invoke it and thus cause funds from the contract to be deposited to the protocol.

While the funds will still be owned by the correct `sellerId`, the `BosonVoucherBase` contract is capable of being the "purchaser" of a conditional offer in the Boson Protocol system. These conditional offers can impose restrictions based on **EIP-20** asset balances that can be compromised by this function in an on-chain race condition.

Impact:

It is presently possible to hijack threshold-based commit authorizations that are performed by the `BosonVoucherBase` by invoking its `BosonVoucherBase::withdrawToProtocol` function.

Example:

contracts/protocol/clients/voucher/BosonVoucher.sol

SOL

```sol
766 function withdrawToProtocol(address[] calldata _tokenList) external {
767     address protocolDiamond =
IClientExternalAddresses(BeaconClientLib._beacon()).getProtocolAddress();
768     uint256 sellerId = getSellerId();
769
770     for (uint256 i = 0; i < _tokenList.length; i++) {
771         address token = _tokenList[i];
772         if (token == address(0)) {
773             uint256 balance = address(this).balance;
774             IBosonFundsHandler(protocolDiamond).depositFunds{ value: balance }
(sellerId, token, balance);
775         } else {
```

Example (Cont.):

```sol
776             uint256 balance = IERC20(token).balanceOf(address(this));
777             IERC20(token).approve(protocolDiamond, balance);
778             IBosonFundsHandler(protocolDiamond).depositFunds(sellerId, token,
balance);
779         }
780     }
781 }
```

Recommendation:

We advise the code to apply proper access control and ensure that the function can only be called by the `OwnableUpgradeable::owner` of the contract.

Alleviation (2b9f60b6c3323fd234b570089ceff924cdb5851c):

The Boson Protocol team evaluated this exhibit and acknowledged that it is an edge condition, however, they do not plan to remediate it as they wish withdrawals to be flexible.

BVR-04M: Inexistent Legacy Compatibility of Boson Voucher Premints

Type	Severity	Location
Logical Fault	Medium	BosonVoucher.sol:L452, L706, L736

Description:

The updated `BosonVoucher` implementation utilizes a new identification system for tokens whereby they are composed of the offer ID as well as exchange ID in the upper and lower halves of the 256 bit number respectively.

Preminted vouchers that had been issued before the update, however, will misbehave as they will not be able to be committed properly and will yield incorrect `BosonVoucher::tokenURI` values.

Impact:

Any `BosonVoucher` beacon implementations that were upgraded and had lingering preminted vouchers will fail to function as expected after the upgrade.

Example:

contracts/protocol/clients/voucher/BosonVoucher.sol

SOL

```sol
693 function _beforeTokenTransfer(address _from, address _to, uint256 _tokenId,
uint256) internal override {
694     // Derive the exchange id
695     uint256 exchangeId = _tokenId & type(uint128).max;
696     if (_isCommitable) {
697         // If is committable, invoke commitToPreMintedOffer on the protocol
698
699         // Set _isCommitable to false
700         _isCommitable = false;
701
702         // Set the preminted token as committed
```

Example (Cont.):

```sol
        _committed[_tokenId] = true;

        // Derive the offer id
        uint256 offerId = _tokenId >> 128;

        // If this is a transfer of preminted token, treat it differently
        address protocolDiamond =
IClientExternalAddresses(BeaconClientLib._beacon()).getProtocolAddress();

IBosonExchangeHandler(protocolDiamond).commitToPreMintedOffer(payable(_to), offerId,
exchangeId);
    } else if (_from != address(0) && _to != address(0) && _from != _to) {
```

Recommendation:

We advise the `BosonVoucher` system to either "delete" all previously issued premints as part of its upgrade or to support them properly, the latter of which we advise given that preminted offers are closely intertwined with offers and the `OrchestrationHandlerFacet1`.

Alleviation (2b9f60b6c3):

The Boson Protocol team stated that they deem this exhibit incorrect as the new identification system was introduced in tandem with the premint functionality of vouchers.

We consider this to be incorrect as the audit we performed of **`v2.2.0-rc.2`** contained a preminted system without the new identification system in place.

We advise the Boson Protocol team to re-evaluate this exhibit and specify what version of `v2.2.0` was actively deployed and will be replaced by the currently-audited version.

Alleviation (584e7d054c):

The Boson Protocol team clarified that the Polygon mainnet deployment has been upgraded from **`v2.2.0`** to **`v2.2.1`** with no intermediate steps and as such, no lingering legacy vouchers are present in the system.

Based on this fact, we consider the exhibit nullified as it arose from a version that was never actively deployed.

BVR-05M: Inexistent Transfer of Preminted Voucher Ranges

Type	Severity	Location
Logical Fault	Medium	BosonVoucher.sol:L175, L227, L291, L483-L489

Description:

The process of issuing a range reservation within `BosonVoucherBase::reserveRange` ensures that the `_to` address is either the contract itself or the contract's owner, however, this assumption may have been invalidated when `BosonVoucherBase::preMint` has been invoked as the transfer of ownership of a voucher (`BosonVoucherBase::transferOwnership`) does not update the range owner values.

Impact:

The current mechanism permits the owner of a voucher to reserve a large range and then transfer their voucher. When doing so, the new owner will be unable to utilize this reserved range for themselves as the owner would have been cached to the previous owner, a behaviour we consider incorrect.

Example:

contracts/protocol/clients/voucher/BosonVoucher.sol

SOL

```sol
152 function reserveRange(uint256 _offerId, uint256 _start, uint256 _length, address
_to) external onlyRole(PROTOCOL) {
153     // _to must be the contract address or the contract owner (operator)
154     require(_to == address(this) || _to == owner(), INVALID_TO_ADDRESS);
155
156     // Prevent reservation of an empty range
157     require(_length > 0, INVALID_RANGE_LENGTH);
158
159     // Adjust start id to include offer id
160     require(_start > 0, INVALID_RANGE_START);
161     _start += (_offerId << 128);
```

Example (Cont.):

```sol

    // Prevent overflow in issueVoucher and preMint
    require(_start <= type(uint256).max - _length, INVALID_RANGE_LENGTH);

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

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

    // Store the reserved range
    range.start = _start;
    range.length = _length;
    range.owner = _to;

    emit RangeReserved(_offerId, range);
}

/**
 * @notice Pre-mints all or part of an offer's reserved vouchers.
 *
 * For small offer quantities, this method may only need to be
 * called once.
 *
 * But, if the range is large, e.g., 10k vouchers, block gas limit
 * could cause the transaction to fail. Thus, in order to support
 * a batched approach to pre-minting an offer's vouchers,
 * this method can be called multiple times, until the whole
```

Example (Cont.):

```sol
 * range is minted.
 *
 * A benefit to the batched approach is that the entire reserved
 * range for an offer need not be pre-minted at one time. A seller
 * could just mint batches periodically, controlling the amount
 * that are available on the market at any given time, e.g.,
 * creating a pre-minted offer with a validity period of one year,
 * causing the token range to be reserved, but only pre-minting
 * a certain amount monthly.
 *
 * Caller must be contract owner (seller assistant address).
 *
 * Reverts if:
 * - Offer id is not associated with a range
 * - Amount to mint is more than remaining un-minted in range
 * - Offer already expired
 * - Offer is voided
 *
 * @param _offerId - the id of the offer
 * @param _amount - the amount to mint
 */
function preMint(uint256 _offerId, uint256 _amount) external onlyOwner {
    // Get the offer's range
    Range storage range = _rangeByOfferId[_offerId];

    // Revert if id not associated with a range
    require(range.length != 0, NO_RESERVED_RANGE_FOR_OFFER);

```

Example (Cont.):

```sol
    // Revert if no more to mint in range
    require(range.length >= range.minted + _amount, INVALID_AMOUNT_TO_MINT);

    // Make sure that offer is not expired or voided
    (Offer memory offer, OfferDates memory offerDates) = getBosonOffer(_offerId);
    require(!offer.voided && (block.timestamp <= offerDates.validUntil),
OFFER_EXPIRED_OR_VOIDED);

    // Get the first token to mint
    uint256 start = range.start + range.minted;
    address to = range.owner;

    // Pre-mint the range
    uint256 tokenId;
    for (uint256 i = 0; i < _amount; i++) {
        tokenId = start + i;

        emit Transfer(address(0), to, tokenId);
    }

    // Bump the minted count
    range.minted += _amount;

    // Update to total balance
    getERC721UpgradeableStorage()._balances[to] += _amount;

    emit VouchersPreMinted(_offerId, start, tokenId);
}
```

Recommendation:

This issue is slightly complex to solve as preminted offer burns should be done on the address that was consumed during a `BosonVoucherBase::preMint` call. As such, we propose a uniform range "ownership" mechanism as follows:

These adjustments to the four premint-related functions are sufficient to alleviate the issue described.

Alleviation (2b9f60b6c3):

The Boson Protocol team stated that they wish to remain compatible with the **EIP-721** standard in full and emit the correct `Transfer` events whenever a premint occurs. As such, they consider the current approach a deliberate choice and wish to acknowledge this exhibit.

We would like to state that this issue should be re-visited by the Boson Protocol team and lead to a potential refactor in the way premints work in the codebase as we consider it a flaw worthy of being remediated due to the potential user-experience hinderances it may cause.

Alleviation (584e7d054c):

After re-evaluating the exhibit, the Boson Protocol has retained their decision to proceed with the current approach for premint operations.

To ensure users of the range functionality are aware of how it behaves, extensive documentation and warnings will be placed in the codebase to note the feature is experimental and detail the caveats that arise from utilizing it.

Based on these actions, we consider this exhibit acknowledged and advise it to be monitored by the Boson Protocol team in future releases of the protocol.

BVR-06M: Insufficient Protection of Contract Assets

Type	Severity	Location
Logical Fault	Major	BosonVoucher.sol:L536-L543

Description:

The `BosonVoucherBase::callExternalContract` is insecure as it permits the contract to perform arbitrary calls, potentially compromising the assets it is in possession of. A short list of disallowed `selector` values has been specified, however, it is insufficient if the contract is expected to not reduce its balance in any way.

Impact:

It is presently possible to compromise funds held within the `BosonVoucher` despite the security measures in place within `BosonVoucherBase::callExternalContract`.

Example:

contracts/protocol/clients/voucher/BosonVoucher.sol

SOL

```
531 function callExternalContract(address _to, bytes calldata _data) external payable
onlyOwner returns (bytes memory) {
532     require(_to != address(0), INVALID_ADDRESS);
533
534     // Prevent invocation of functions that would allow transfer of tokens from
this contract
535     bytes4 selector = bytes4(_data[:4]);
536     require(
537         selector != IERC20.transfer.selector &&
538             selector != IERC20.approve.selector &&
539             selector != IERC20.transferFrom.selector &&
540             selector != DAI.push.selector &&
```

Example (Cont.):

```sol
            selector != DAI.move.selector,
        FUNCTION_NOT_ALLOWLISTED
    );

    return _to.functionCallWithValue(_data, msg.value,
FUNCTION_CALL_NOT_SUCCESSFUL);
}
```

Recommendation:

We advise either the `selector` list to be made a `mapping` that can be maintained by the Boson Protocol team as the current solution is insufficient in preventing token transfers.

As tangible examples, an `ERC20::burn` call can be made on the `DAI` token, an `ERC20Like::increaseAllowance` call can be made on the `USDC` token, and `ERC20Like::increaseApproval` as well as `ERC20Like::increaseAllowance` functions exist on multiple tokens.

Depending on the intended purpose of the **`BosonVoucherBase::callExternalContract`** function, multiple robust solutions can be employed. We will detail a few approaches in brief that the Boson Protocol can follow below:

To note, the contract seems to solely protect **EIP-20** assets. If **EIP-721** or **EIP-1155** assets need to be protected as well (which we believe to be the case), all the aforementioned solutions would need to be adapted accordingly to support those two standards on top (i.e. `ERC721::ownerOf` instead of `ERC20::balanceOf`).

Alleviation (2b9f60b6c3323fd234b570089ceff924cdb5851c):

The Boson Protocol proceeded with applying the second approach we detailed whereby an `IERC20::balanceOf` invocation is attempted on the target `_to` address and if it succeeds the interaction is prohibited.

The Boson Protocol team stated that they do not wish to support **EIP-721** / **EIP-1155** asset protection at this stage, however, it is something that they will keep track of for a potential future implementation.

Given that **EIP-20** assets are adequately protected via the remediation introduced, we consider this exhibit to be fully alleviated.

BVR-07M: Storage Conflict of Beacon Implementation

Type	Severity	Location
Language Specific	Major	BosonVoucher.sol:L51-L54

Description:

The `BosonVoucherBase` contract is meant to be the logic target of beacon implementations per its documentation as well as the project's code structure, however, in the latest update the `_rangeOfferIds` entry was omitted thereby shifting its storage space upwards by one 32-byte slot and thus rendering the contract an incompatible upgrade for existing beacons.

Impact:

If the updated `BosonVoucherBase` was utilized as a beacon upgrade of the existing Boson Protocol, the storage space of all deployed vouchers would be corrupted causing significant unintended side effects.

Example:

contracts/protocol/clients/voucher/BosonVoucher.sol

SOL

```sol
// Map an offerId to a Range for pre-minted offers
mapping(uint256 => Range) private _rangeByOfferId;

// Premint status, used only temporarly in transfers
bool private _isCommitable;
```

Recommendation:

We advise the code to retain the `_rangeOfferIds` as a `private` variable that is aptly renamed to `_rangeOfferIds_deprecated` so as to indicate that it solely exists to retain the storage offset of the overall system.

Alleviation (2b9f60b6c3323fd234b570089ceff924cdb5851c):

The Boson Protocol team stated that the `_rangeOfferIds` variable was present in a release candidate that never made it to production.

As such, we consider this exhibit nullified given that the storage layout of the live contracts is unaffected by the referenced discrepancy.

BundleBase Manual Review Findings

BBS-01M: Removal of Bundle Limitations

Type	Severity	Location
Input Sanitization	Medium	BundleBase.sol:L48-L54

Description:

The protocol-level limitations for the `BundleBase::createBundleInternal` function have been removed whereas they should remain per the discussions of the Boson Protocol team in the relevant **PR#675** due to the valid concerns raised in the referenced thread.

Namely, a significantly high number of `twinIds` per bundle can cause the transfer of twins to reach the block gas limit and thus cause a Denial-of-Service attack rendering a voucher unredeemable.

Impact:

The current `BundleBase::createBundleInternal` function will permit the creation of a bundle that can result in a Denial-of-Service attack when redeeming a voucher associated with the bundle.

Example:

contracts/protocol/bases/BundleBase.sol

SOL

```sol
36  function createBundleInternal(Bundle memory _bundle) internal {
37      // Cache protocol lookups and limits for reference
38      ProtocolLib.ProtocolLookups storage lookups = protocolLookups();
39
40      // get message sender
41      address sender = msgSender();
42
43      // get seller id, make sure it exists and store it to incoming struct
44      (bool exists, uint256 sellerId) = getSellerIdByAssistant(sender);
45      require(exists, NOT_ASSISTANT);
```

Example (Cont.):

```sol

    // validate that offer ids and twin ids are not empty
    require(
        _bundle.offerIds.length > 0 && _bundle.twinIds.length > 0,
        BUNDLE_REQUIRES_AT_LEAST_ONE_TWIN_AND_ONE_OFFER
    );

    // Get the next bundle and increment the counter
    uint256 bundleId = protocolCounters().nextBundleId++;
```

Recommendation:

We advise these limitations to be re-imposed, ensuring that the protocol's functions cannot be maliciously invoked to create an unredeemable voucher.

Alleviation (2b9f60b6c3323fd234b570089ceff924cdb5851c):

The Boson Protocol team evaluated this exhibit and maintained that a hard-limit on the total twin IDs attached to a bundle is no longer needed as the code will mark a twin transfer as failed when the code is approaching its gas limit.

We would like to denote that it may still be possible to perform a denial-of-service attack with an abnormally high amount of twin IDs as the gas required to evaluate the normal statements of the loop can consume the `MINIMAL_RESIDUAL_GAS`. In any case, we consider this exhibit to be nullified as the original denial-of-service described is unfeasible.

ConfigHandlerFacet Manual Review Findings

CHF-01M: Inexistent Validation of Proper Resolution Period Configuration

Type	Severity	Location
Input Sanitization	Minor	ConfigHandlerFacet.sol:L427-L433, L455-L461

Description:

The `ConfigHandlerFacet::setMinResolutionPeriod` and `ConfigHandlerFacet::setMaxResolutionPeriod` functions do not properly sanitize their input arguments as conforming to their counterpart `maxResolutionPeriod` and `minResolutionPeriod` respectively.

Impact:

It is presently possible to misconfigure the resolution period range of the protocol limits which is an undesirable trait of the `ConfigHandlerFacet` contract.

Example:

contracts/protocol/facets/ConfigHandlerFacet.sol

SOL

```
416 /**
417  * @notice Sets the minimum resolution period a seller can specify.
418  *
419  * Emits a MinResolutionPeriodChanged event.
420  *
421  * Reverts if _minResolutionPeriod is zero.
422  *
423  * @dev Caller must have ADMIN role.
424  *
425  * @param _minResolutionPeriod - the minimum resolution period that a
{BosonTypes.Seller} can specify
```

Example (Cont.):

```sol
 */
function setMinResolutionPeriod(uint256 _minResolutionPeriod) public override onlyRole(ADMIN) nonReentrant {
    // Make sure _maxResolutionPeriod is greater than 0
    checkNonZero(_minResolutionPeriod);

    protocolLimits().minResolutionPeriod = _minResolutionPeriod;
    emit MinResolutionPeriodChanged(_minResolutionPeriod, msgSender());
}

/**
 * @notice Gets the minimum resolution period a seller can specify.
 *
 * @return the minimum resolution period that a {BosonTypes.Seller} can specify
 */
function getMinResolutionPeriod() external view override returns (uint256) {
    return protocolLimits().minResolutionPeriod;
}

/**
 * @notice Sets the maximum resolution period a seller can specify.
 *
 * Emits a MaxResolutionPeriodChanged event if successful.
 *
 * Reverts if the _maxResolutionPeriod is zero.
 *
 * @dev Caller must have ADMIN role.
 *
 * @param _maxResolutionPeriod - the maximum resolution period that a {BosonTypes.Seller} can specify
```

Example (Cont.):

```sol
454  */
455 function setMaxResolutionPeriod(uint256 _maxResolutionPeriod) public override
onlyRole(ADMIN) nonReentrant {
456     // Make sure _maxResolutionPeriod is greater than 0
457     checkNonZero(_maxResolutionPeriod);
458
459     protocolLimits().maxResolutionPeriod = _maxResolutionPeriod;
460     emit MaxResolutionPeriodChanged(_maxResolutionPeriod, msgSender());
461 }
```

Recommendation:

We advise each function to ensure that the value is greater-than or less-than its counterpart, guaranteeing that the `minResolutionPeriod` and `maxResolutionPeriod` values of `protocolLimits()` yield a correctly defined range of values.

Alleviation (2b9f60b6c3323fd234b570089ceff924cdb5851c):

Both functions have been updated to ensure the minimum and maximum resolution period limitations define a valid range of values, alleviating this exhibit.

ExchangeHandlerFacet Manual Review Findings

EHF-01M: Breaking Change of Single-Point-of-Entry

Type	Severity	Location
Logical Fault	Unknown	ExchangeHandlerFacet.sol:L76, L127

Description:

The `ExchangeHandlerFacet::commitToOffer` function was the single point of entry of the particular facet for committing to offers within the Boson Protocol system.

A new feature around enhanced token gating split the functionality of committing to offers to two implementations; `ExchangeHandlerFacet::commitToOffer` and `ExchangeHandlerFacet::commitToConditionalOffer`. The issue with the original implementation is that it will currently `revert` if the commit cannot occur due to the offer being part of a group with a condition.

Impact:

The severity of this exhibit depends on the current adoption of the Boson Protocol system and whether the Boson Protocol team has reached out to these parties to verify that the upgrade is compatible with their systems.

Example:

contracts/protocol/facets/ExchangeHandlerFacet.sol

SOL

```sol
76  function commitToOffer(
77      address payable _buyer,
78      uint256 _offerId
79  ) external payable override exchangesNotPaused buyersNotPaused nonReentrant {
80      // Make sure buyer address is not zero address
81      require(_buyer != address(0), INVALID_ADDRESS);
82
83      Offer storage offer = getValidOffer(_offerId);
84
85      // For there to be a condition, there must be a group.
```

Example (Cont.):

```sol
86      (bool exists, uint256 groupId) = getGroupIdByOffer(offer.id);
87      if (exists) {
88          // Get the condition
89          Condition storage condition = fetchCondition(groupId);
90
91          // Make sure group doesn't have a condition. If it does, use
commitToConditionalOffer instead.
92          require(condition.method == EvaluationMethod.None, GROUP_HAS_CONDITION);
93      }
94
95      commitToOfferInternal(_buyer, offer, 0, false);
96  }
```

Recommendation:

As the Boson Protocol system has most likely been integrated by external parties, these external parties will no longer be able to interact with conditional offers via the `ExchangeHandlerFacet::commitToOffer` function that may lead to significant consequences depending on whether those parties possess upgradeable code or not.

We advise the `ExchangeHandlerFacet::commitToOffer` function to redirect its call to `ExchangeHandlerFacet::commitToConditionalOffer` with a `_tokenId` of `0`, permitting a variety of conditions to still be satisfied.

In general, the new per-token-ID-commit token gating feature should be backwards-compatible with the original restriction system and assume default values for the `_tokenId` if it is unspecified.

Alleviation (2b9f60b6c3):

The Boson Protocol team stated that they do not wish to expose the same functionality across two different functions and that they consider the change safe to apply to the system in its current state.

We would like to state that the Boson Protocol team has not confirmed that integrators of their systems will adequately react to this non-backwards-compatible change and we advise the Boson Protocol team to proactively reach out to potentially affected parties rather than dealing with them after-the-fact.

Alleviation (584e7d054c):

The Boson Protocol team re-evaluated this exhibit and stated that the sole integrators of their system at its current state are internal and have been prepared for this change.

As such, we consider this exhibit nullified.

EHF-02M: Restrictive Migration Mechanism

Type	Severity	Location
Language Specific	Informational	ExchangeHandlerFacet.sol:L38

Description:

The `ExchangeHandlerFacet` introduces an upgrade to the `BosonVoucher` identification system, however, the way the `ExchangeHandlerFacet` differentiates between the two is based on a "snapshot" `EXCHANGE_ID_2_2_0` beyond which IDs are assumed to be of the new representation format.

Impact:

This exhibit acts as a warning of a potential caveat of the `EXCHANGE_ID_2_2_0` system that the Boson Protocol may not be aware of as well as a way to make the distinction between legacy and new-style IDs more robust.

Example:

contracts/protocol/facets/ExchangeHandlerFacet.sol

SOL

```sol
37  constructor(uint256 _firstExchangeId2_2_0) {
38      EXCHANGE_ID_2_2_0 = _firstExchangeId2_2_0;
39  }
```

Recommendation:

While this approach is correct, it will misbehave if the `ExchangeHandlerFacet` and `BosonVoucher` implementations are not upgraded properly.

Namely, the exchange and buyer regions of the protocol will have to be paused before the `BosonVoucher` implementations are upgraded as otherwise vouchers may be issued between a protocol update and its execution, causing the minted IDs to be improperly parsed.

Alternatively, the code could differentiate between the new and legacy type IDs by evaluating the upper 128 bits of the ID. If the upper 128 bits are zero, then it is of the legacy type as a zero-value offer ID is not possible.

Alleviation (2b9f60b6c3323fd234b570089ceff924cdb5851c):

The Boson Protocol team stated that they will apply proper upgrade procedures to ensure that the ID system does not become corrupt in-between upgrades.

Furthermore, they stated that a legacy ID evaluation system was experimented but was deemed to complex and gas-expensive for a production environment.

As such, we consider this exhibit alleviated **based on the fact that the Boson Protocol team will follow proper upgrade procedures for their contracts**.

EHF-03M: Bypass of Token Specific Conditions

Type	Severity	Location
Logical Fault	Major	ExchangeHandlerFacet.sol:L148-L154, L156-L158

Description:

As par of the condition overhaul supporting range-based token gating with limited commits per unique token ID, the `ExchangeHandlerFacet::authorizeCommit` function takes an extra `_tokenId` argument that is not validated whereas previously it was utilizing the `_condition.tokenId`.

While the `ExchangeHandlerFacet::commitToPreMintedOffer` function will automatically utilize the correct `tokenId` value based on the `condition`, the `ExchangeHandlerFacet::commitToConditionalOffer` does not perform this check. As such, it is possible for a group that is meant to be satisfied by a specific NFT ID to be satisfied by any.

Impact:

A `Condition` that is `MultiToken` and has a `Threshold` or that is a `SpecificToken` condition will not evaluate properly in the `ExchangeHandlerFacet::commitToConditionalOffer` function in the current iteration of the codebase and can be trivially bypassed.

Example:

contracts/protocol/facets/ExchangeHandlerFacet.sol

SOL

```sol
127 function commitToConditionalOffer(
128     address payable _buyer,
129     uint256 _offerId,
130     uint256 _tokenId
131 ) external payable override exchangesNotPaused buyersNotPaused nonReentrant {
132     // Make sure buyer address is not zero address
133     require(_buyer != address(0), INVALID_ADDRESS);
134
135     Offer storage offer = getValidOffer(_offerId);
136
```

Example (Cont.):

```sol
    // For there to be a condition, there must be a group.
    (bool exists, uint256 groupId) = getGroupIdByOffer(offer.id);

    // Make sure the group exists
    require(exists, NO_SUCH_GROUP);

    // Get the condition
    Condition storage condition = fetchCondition(groupId);

    require(condition.method != EvaluationMethod.None, GROUP_HAS_NO_CONDITION);

    if (condition.length >= 1) {
        // If condition has length >= 1, check that the token id is in range
        require(
            _tokenId >= condition.tokenId && _tokenId < condition.tokenId + condition.length,
            TOKEN_ID_NOT_IN_CONDITION_RANGE
        );
    }

    if (condition.method == EvaluationMethod.Threshold && condition.tokenType != TokenType.MultiToken) {
        require(_tokenId == 0, INVALID_TOKEN_ID);
    }

    bool allow = authorizeCommit(_buyer, condition, groupId, _tokenId);
```

Recommendation:

We advise the code of `ExchangeHandlerFacet::commitToConditionalOffer` function to be updated, either ensuring that the input `_tokenId` has been correctly specified or overwriting it with the correct value based on the `condition` of the group being evaluated.

Alleviation (2b9f60b6c3323fd234b570089ceff924cdb5851c):

The code of `ExchangeHandlerFacet::commitToConditionalOffer` was updated to properly evaluate whether a `SpecificToken` evaluation method or a token type of `MultiToken` has been specified and to validate the provided `_tokenId` in such a case.

As such, we consider this exhibit fully alleviated.

GroupBase Manual Review Findings

GBE-01M: Insufficient Validation of Conditions

Type	Severity	Location
Input Sanitization	Medium	GroupBase.sol:L139-L142, L150-L156

Description:

The `GroupBase::validateCondition` function insufficiently sanitizes its input `Condition`. Specifically, it fails to ensure that a `tokenId` has been specified in the following cases:

Impact:

It is presently possible to validate conditions that should otherwise fail and could cause the condition-consumption mechanisms of other Boson Protocol modules to misbehave.

Example:

contracts/protocol/bases/GroupBase.sol

SOL

```sol
120 function validateCondition(Condition calldata _condition) internal pure returns
(bool) {
121     bool valid = true;
122     if (_condition.method == EvaluationMethod.None) {
123         valid = (_condition.tokenAddress == address(0) &&
124             _condition.tokenId == 0 &&
125             _condition.threshold == 0 &&
126             _condition.maxCommits == 0 &&
127             _condition.length == 0);
128     } else {
129         if (_condition.tokenId != 0) {
```

Example (Cont.):

```sol
            valid = _condition.length != 0;
            valid = valid && type(uint256).max - _condition.length >= _condition.tokenId;
        }

        if (_condition.method == EvaluationMethod.Threshold) {
            valid =
                valid &&
                (_condition.tokenAddress != address(0) && _condition.maxCommits > 0 && _condition.threshold > 0);

            if (_condition.tokenType != TokenType.MultiToken) {
                // NonFungibleToken and FungibleToken should not have length and tokenId
                valid = valid && _condition.length == 0 && _condition.tokenId == 0;
            }
        } else {
            valid =
                valid &&
                (_condition.tokenAddress != address(0) &&
                    _condition.maxCommits > 0 &&
                    _condition.tokenType != TokenType.FungibleToken); // FungibleToken not allowed for SpecificToken

            // SpecificToken with NonFungibleToken should not have threshold
            if (_condition.tokenType == TokenType.NonFungibleToken) {
                valid = valid && _condition.threshold == 0;
            } else {
                // SpecificToken with MultiToken should have threshold
                valid = valid && _condition.threshold > 0;
            }
        }
```

Example (Cont.):

```sol
    }

    return valid;
}
```

Recommendation:

We advise the code to properly guarantee that a `tokenId` has been specified in the aforementioned combination of cases as they comprise improperly defined conditions that should not be validated.

Alleviation (2b9f60b6c3323fd234b570089ceff924cdb5851c):

The Boson Protocol team evaluated this exhibit and stated that a given `tokenId` of `0` is valid and as such, the recommended course of action cannot be applied.

We concur with this assessment and thus consider this exhibit to be nullified.

PauseHandlerFacet Manual Review Findings

PHF-01M: Incorrect Revert Condition Specification

Type	Severity	Location
Code Style	Minor	PauseHandlerFacet.sol:L33, L52, L109

Description:

The referenced comments insinuate that the `PauseHandlerFacet::pause`, `PauseHandlerFacet::unpause`, and `PauseHandlerFacet::togglePause` functions will `revert` if a duplicate region is specified, however, the code itself makes sure to calculate the correct state value even if duplicate regions exist.

Impact:

Callers of the `PauseHandlerFacet` functions may assume that their regions are guaranteed to be unique when that is not the case.

Example:

contracts/protocol/facets/PauseHandlerFacet.sol

SOL

```
103 /**
104  * @notice Toggles pause/unpause for some or all of the protocol.
105  *
106  * Toggle all regions if none are specified.
107  *
108  * Reverts if:
109  * - A region is specified more than once
110  *
111  * @param _regions - an array of regions to pause/unpause. See:
{BosonTypes.PausableRegion}
112  * @param _pause - a boolean indicating whether to pause (true) or unpause
(false)
```

Example (Cont.):

```sol
 */
function togglePause(BosonTypes.PausableRegion[] calldata _regions, bool _pause) internal {
    // Cache protocol status for reference
    ProtocolLib.ProtocolStatus storage status = protocolStatus();

    // Toggle all regions if none are specified.
    if (_regions.length == 0) {
        // Store the toggle scenario
        status.pauseScenario = _pause ? ALL_REGIONS_MASK : 0;
        return;
    }

    uint256 region;
    uint256 incomingScenario;

    // Calculate the incoming scenario as the sum of individual regions
    // Use "or" to get the correct value even if the same region is specified more than once
    for (uint256 i = 0; i < _regions.length; i++) {
        // Get enum value as power of 2
        region = 1 << uint256(_regions[i]);
        incomingScenario |= region;
    }

    // Store the toggle scenario
    if (_pause) {
        // for pausing, just "or" the incoming scenario with the existing one
        // equivalent to summation
        status.pauseScenario |= incomingScenario;
```

Example (Cont.):

```sol
    } else {
        // for unpausing, "and" the inverse of the incoming scenario with the existing one
        // equivalent to subtraction
        status.pauseScenario &= ~incomingScenario;
    }
}
```

Recommendation:

We advise the code to either prevent duplicate regions or remove the `revert` condition from the documentation, either of which we consider an adequate resolution and the latter of which we assume to be appropriate action based on the `PauseHandlerFacet::togglePause` function's implementation.

Alleviation (2b9f60b6c3323fd234b570089ceff924cdb5851c):

The documentation was corrected by the Boson Protocol team rendering this exhibit alleviated.

ProtocolInitializationHandlerFacet Manual Review Findings

PIH-01M: Inexistent Support of Manual Seller Configuration

Type	Severity	Location
Language Specific	Minor	ProtocolInitializationHandlerFacet.sol:L172-L178

Description:

The `ProtocolInitializationHandlerFacet::initV2_3_0` function will assign the `sellerCreator` values of multiple seller IDs to ensure that existing deployed seller IDs remain compatible with the updated system, however, this type of upgrade mechanism can only process a finite number of seller IDs and can become inexecutable in the future.

Impact:

The current v2.3.0 upgrade initialization methodology may become unsustainable as the protocol grows due to the inherent block gas limits that are present on all blockchains.

Example:

contracts/protocol/facets/ProtocolInitializationHandlerFacet.sol

SOL

```sol
172 for (uint256 i = 0; i < sellerIds.length; i++) {
173     (bool exists, , ) = fetchSeller(sellerIds[i]);
174     require(exists, NO_SUCH_SELLER);
175     require(sellerCreators[i] != address(0), INVALID_ADDRESS);
176
177     lookups.sellerCreator[sellerIds[i]] = sellerCreators[i];
178 }
```

Recommendation:

We advise the code to instead allow `sellerCreator` values to be filled in either via administrative action in batches, or by the owner of a particular `BosonVoucher` instance, either of which we consider an adequate remediation to this exhibit.

Alleviation (2b9f60b6c3323fd234b570089ceff924cdb5851c):

Initialization of seller IDs has been eliminated from the `ProtocolInitializationHandlerFacet::initV2_3_0` function as a different methodology of associating sellers with their voucher clones has been employed in the form of seller "salts".

These salts are auto-filled for accounts created before the `v2.3.0` deployment of the Boson Protocol, rendering this exhibit alleviated as the unsustainable seller ID-to-creator association is no longer performed in the protocol's initialization.

PIH-02M: Insufficient Sanitization of Minimum Resolution Period

Type	Severity	Location
Input Sanitization	Minor	ProtocolInitializationHandlerFacet.sol:L165

Description:

The `_minResolutionPeriod` that is newly specified in the **`ProtocolInitializationHandlerFacet::initV2_3_0`** function is inadequately sanitized as it may be greater than the current `maxResolutionPeriod`.

Impact:

The contract can presently be misconfigured during its deployment which should be prohibited.

Example:

contracts/protocol/facets/ProtocolInitializationHandlerFacet.sol

SOL

```
164 // Initialize limits.maxPremintedVouchers (configHandlerFacet initializer)
165 require(_minResolutionPeriod != 0, VALUE_ZERO_NOT_ALLOWED);
166 protocolLimits().minResolutionPeriod = _minResolutionPeriod;
167 emit MinResolutionPeriodChanged(_minResolutionPeriod, msgSender());
```

Recommendation:

We advise the `require` check to either be expanded or a new one to be introduced ensuring that the value of `_minResolutionPeriod` is less-than the value of the current `protocolLimits().maxResolutionPeriod`, preventing misconfiguration of the contract during its initialization.

Alleviation (2b9f60b6c3323fd234b570089ceff924cdb5851c):

The minimum resolution period is adequately sanitized per our recommendation in the **`ProtocolInitializationHandlerFacet::initV2_3_0`** function, alleviating this exhibit.

SellerBase Manual Review Findings

SBE-01M: Restrictive Deployment of Voucher Clone

Type	Severity	Location
Logical Fault	Minor	SellerBase.sol:L99, L172

Description:

The `SellerBase::createSellerInternal` function will deploy a `BosonVoucher` instance utilizing the `create2` deployment methodology and a salt derived from the `sender` as well as an external ID value.

This external ID value is hard-coded to an empty string (`""`) during the creation of a seller, meaning that its `sender` can deploy a `BosonVoucher` once. This type of functionality significantly restricts the flexibility offered to users as well as integrators of the Boson Protocol system; especially automated seller deployers.

Impact:

The impact of this change may be severe depending on the adoption of the Boson Protocol system and how many automated smart contracts utilize the `SellerBase::createSellerInternal` workflows (such as `SellerHandlerFacet::createSeller` or the `OrchestrationHandlerFacet1` contract's `createSeller` prefixed functions).

Example:

contracts/protocol/bases/SellerBase.sol

SOL

```
98  // Create clone and store its address cloneAddress
99  address voucherCloneAddress = cloneBosonVoucher(sellerId, 0, sender,
_seller.assistant, _voucherInitValues, "");
```

Recommendation:

We advise the code to re-evaluate its approach in regard to `create2` deployments, potentially utilizing two types of "ID" values for the `salt` generation; one for the "upper-level" `BosonVoucher` seller ID deployment and one "lower-level" for each collection deployment.

Alleviation (2b9f60b6c3323fd234b570089ceff924cdb5851c):

A specialized collection-level salt is now utilized that can be "reserved" by an address and utilized for the deployment of a collection-level Boson Voucher.

As the codebase has been significantly refactored to accommodate for this change and it is possible to have multiple Boson Voucher deployments depending on the collections desired, we consider this exhibit fully alleviated.

SBE-02M: Incorrect Association of Seller ID

Type	Severity	Location
Logical Fault	Medium	SellerBase.sol:L145

Description:

The `SellerBase::createSellerInternal` function will permanently associate a seller ID with its creator even if the voucher generated from it is transferred.

As such, the new owner will be able to deploy collection-level `BosonVoucher` instances utilizing `salt` values that would have been generated by the original deployer.

Impact:

As the creator of a seller ID is permanently associated with their voucher, the new owner of a voucher will be able to "hijack" the original owner's collection-level Boson Voucher deployments by utilizing the same external ID.

Additionally, a particular user may have multiple collections deployed on chain A and none deployed on chain B; a user who legitimately acquires ownership of the `BosonVoucher` on chain B will be able to deploy collection-level `BosonVoucher` instances on the same addresses as chain A even if the original owner did not deploy them.

Example:

contracts/protocol/bases/SellerBase.sol

SOL

```
145 _lookups.sellerCreator[_seller.id] = msgSender();
```

Recommendation:

We advise this issue to be alleviated in tandem with the "Restrictive Deployment of Voucher Clone" exhibit as it relates to how `salt` values are generated within **`SellerBase::cloneBosonVoucher`**.

Alleviation (2b9f60b6c3):

The code was updated to utilize a seller ID salt system that is also update-able, meaning that it is possible to disassociate a seller ID with its previous salt and update it to point to a new one.

We would like to note that we consider this exhibit partially alleviated as the seller salt that was previously "reserved" should remain so as it has been used in the past. To address this, we advise the `false` assignment in the newly declared `SellerHandlerFacet::updateSellerSalt` function to be omitted.

Alleviation (584e7d054c):

After extensive discussions with the Boson Protocol team, we concluded that the risk arising from salt re-use is negligible and the flexibility permitted by it far outweighs it.

As such, we consider this exhibit fully alleviated per the original remediation.

SellerHandlerFacet Manual Review Findings

SHF-01M: Incorrect Iterator Usage

Type	Severity	Location
Logical Fault	Major	SellerHandlerFacet.sol:L277

Description:

The nested `for` loop within the `SellerHandlerFacet::optInToSellerUpdate` function re-uses the `i` variable of the outer loop instead of declaring a new one. As such, the inner loop can corrupt the outer loop's iteration.

Impact:

If the `SellerHandlerFacet::optInToSellerUpdate` function is invoked with multiple `_fieldsToUpdate`, the iterator will potentially skip or repeat fields to update as it will jump from `i` to `collectionCount`, the latter of which can be greater than or lower than the expected value of `i + 1`.

Example:

contracts/protocol/facets/SellerHandlerFacet.sol

SOL

```sol
233 for (uint256 i = 0; i < _fieldsToUpdate.length; i++) {
234     SellerUpdateFields role = _fieldsToUpdate[i];
235
236     // Approve admin update
237     if (role == SellerUpdateFields.Admin && sellerPendingUpdate.admin !=
address(0)) {
238         require(sellerPendingUpdate.admin == sender, UNAUTHORIZED_CALLER_UPDATE);
239
240         preUpdateSellerCheck(_sellerId, sender, lookups);
241
242         // Delete old seller id by admin mapping
```

Example (Cont.):

```sol
        delete lookups.sellerIdByAdmin[seller.admin];

        // Update admin
        seller.admin = sender;

        // Store new seller id by admin mapping
        lookups.sellerIdByAdmin[sender] = _sellerId;

        // Delete pending update admin
        delete sellerPendingUpdate.admin;

        // Delete auth token for seller id if it exists
        if (authToken.tokenType != AuthTokenType.None) {
            delete lookups.sellerIdByAuthToken[authToken.tokenType]
[authToken.tokenId];
            delete protocolEntities().authTokens[_sellerId];
        }

        updateApplied = true;
    } else if (role == SellerUpdateFields.Assistant &&
sellerPendingUpdate.assistant != address(0)) {
        // Approve assistant update
        require(sellerPendingUpdate.assistant == sender,
UNAUTHORIZED_CALLER_UPDATE);

        preUpdateSellerCheck(_sellerId, sender, lookups);

        // Delete old seller id by assistant mapping
        delete lookups.sellerIdByAssistant[seller.assistant];

        // Update assistant
```

Example (Cont.):

```sol
271         seller.assistant = sender;
272
273         // Transfer ownership of voucher contract to new assistant
274         IBosonVoucher(lookups.cloneAddress[_sellerId]).transferOwnership(sender);
// default voucher contract
275         Collection[] storage sellersAdditionalCollections =
lookups.additionalCollections[_sellerId];
276         uint256 collectionCount = sellersAdditionalCollections.length;
277         for (i = 0; i < collectionCount; i++) {
278             // Additional collections (if they exist)
279
IBosonVoucher(sellersAdditionalCollections[i].collectionAddress).transferOwnership(se
nder);
280         }
```

Recommendation:

We advise a new iterator to be declared (i.e. `j`) to ensure that the loops do not affect each other's iterators.

Alleviation (2b9f60b6c3323fd234b570089ceff924cdb5851c):

The inner loop iterator has been correctly updated to a new variable labelled `j`, alleviating this exhibit.

BosonVoucher Code Style Findings

BVR-01C: Potential Legibility Optimization

Type	Severity	Location
Code Style	Informational	BosonVoucher.sol:L76

Description:

The voucher identification system of `BosonVoucherBase::initializeVoucher` has been updated to incorporate both the seller ID and the collection index, separating them with an underscore.

Example:

contracts/protocol/clients/voucher/BosonVoucher.sol

SOL

```
76  string memory sellerId = string.concat(Strings.toString(_sellerId), "_",
Strings.toString(_collectionIndex));
```

Recommendation:

We advise an `S` prefix to be introduced to the `_sellerId` and a `C` prefix to be introduced to the `_collectionIndex`, increasing the legibility of the new identification system and allowing readers to immediately discern between the two numerical values without needing to memorize their order.

Alleviation (2b9f60b6c3323fd234b570089ceff924cdb5851c):

Our recommendation was applied properly, discerning between the seller ID and collection indexes with relevant capitalized-letter prefixes.

BVR-02C: Redundant Parenthesis Statements

Type	Severity	Location
Code Style	Informational	BosonVoucher.sol:L113, L223, L279, L282, L317

Description:

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

Example:

contracts/protocol/clients/voucher/BosonVoucher.sol

SOL

```
113 require((_tokenId < rangeStart) || (_tokenId >= rangeStart + range.length),
EXCHANGE_ID_IN_RESERVED_RANGE);
```

Recommendation:

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

Alleviation (2b9f60b6c3323fd234b570089ceff924cdb5851c):

All redundant parenthesis statements have been safely omitted.

ExchangeHandlerFacet Code Style Findings

EHF-01C: Inefficient `mapping` Lookups

Type	Severity	Location
Gas Optimization	Informational	ExchangeHandlerFacet.sol:L993, L1001, L1005, L1013

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/ExchangeHandlerFacet.sol

SOL

```
991 if (_condition.method == EvaluationMethod.SpecificToken) {
992     // How many times has this token id been used to commit to offers in the
group?
993     uint256 commitCount = lookups.conditionalCommitsByTokenId[_tokenId][_groupId];
994
995     require(commitCount < _condition.maxCommits, MAX_COMMITS_TOKEN_REACHED);
996
997     allow = holdsSpecificToken(_buyer, _condition, _tokenId);
998
999     if (allow) {
100
0           // Increment number of commits to the group for this token id if they are
allowed to commit
```

Example (Cont.):

SOL

```
            lookups.conditionalCommitsByTokenId[_tokenId][_groupId] = ++commitCount;
        }
    } else if (_condition.method == EvaluationMethod.Threshold) {
        // How many times has this address committed to offers in the group?
        uint256 commitCount = lookups.conditionalCommitsByAddress[_buyer][_groupId];

        require(commitCount < _condition.maxCommits, MAX_COMMITS_ADDRESS_REACHED);

        allow = holdsThreshold(_buyer, _condition, _tokenId);

        if (allow) {
            // Increment number of commits to the group for this address if they are allowed to commit
            lookups.conditionalCommitsByAddress[_buyer][_groupId] = ++commitCount;
        }
    } else {
```

Recommendation:

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

Alleviation (2b9f60b6c3323fd234b570089ceff924cdb5851c):

All highlighted `mapping` lookups have been adequately optimized, addressing this exhibit.

EHF-02C: Redundant Parenthesis Statement

Type	Severity	Location
Code Style	Informational	ExchangeHandlerFacet.sol:L1065

Description:

The referenced statement is redundantly wrapped in parenthesis (`()`).

Example:

contracts/protocol/facets/ExchangeHandlerFacet.sol

SOL

```
106
5   return (IERC721(_condition.tokenAddress).ownerOf(_tokenId) == _buyer);
```

Recommendation:

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

Alleviation (2b9f60b6c3):

The redundant parenthesis statement was removed in an interim PR, however, the final commit still contains it rendering this exhibit "partially" alleviated.

Alleviation (584e7d054c):

The redundant parenthesis statement has been properly removed in the latest commit hash evaluated by the audit, rendering this exhibit alleviated.

FundsHandlerFacet Code Style Findings

FHF-01C: Loop Iterator Optimizations

Type	Severity	Location
Gas Optimization	Informational	FundsHandlerFacet.sol:L186, L224

Description:

The linked `for` loops increment / decrement their iterator "safely" due to Solidity's built - in safe arithmetics (post-`0.8.X`).

Example:

contracts/protocol/facets/FundsHandlerFacet.sol

SOL

```sol
186 for (uint i = 0; i < _limit; i++) {
```

Recommendation:

We advise the increment / decrement operations to be performed in an `unchecked` code block as the last statement within each `for` loop to optimize their execution cost.

Alleviation (2b9f60b6c3323fd234b570089ceff924cdb5851c):

The referenced loop iterator increment statements have been properly relocated to the end of each `for` loop's body and wrapped in an `unchecked` code block, optimizing their execution cost.

OfferBase Code Style Findings

OBE-01C: Illegible Representation of Limitation

Type	Severity	Location
Code Style	Informational	OfferBase.sol:L327

Description:

The referenced limitation for the `_length` variable is performed by ensuring it is less-than the value of `1 << 64`.

Example:

contracts/protocol/bases/OfferBase.sol

SOL

```
327 require(_length < (1 << 64), INVALID_RANGE_LENGTH);
```

Recommendation:

We advise the code to be adjusted to instead evaluate that the `_length` value is less-than-or-equal-to the value of `type(uint64).max`, greatly enhancing the legibility of the `require` check whilst retaining the same logical constraint.

Alleviation (2b9f60b6c3323fd234b570089ceff924cdb5851c):

The `_length` limitation referenced was updated to properly utilize the `type(uint64).max` compiler literal and to use a less-than-or-equal-to (`<=`) comparator, retaining the same gas cost as the original code whilst increasing its legibility greatly.

PauseHandlerFacet Code Style Findings

PHF-01C: Loop Iterator Optimizations

Type	Severity	Location
Gas Optimization	Informational	PauseHandlerFacet.sol:L81, L87, L130

Description:

The linked `for` loops increment / decrement their iterator "safely" due to Solidity's built - in safe arithmetics (post-`0.8.X`).

Example:

contracts/protocol/facets/PauseHandlerFacet.sol

SOL

```sol
81  for (uint256 i = 0; i < totalRegions; i++) {
```

Recommendation:

We advise the increment / decrement operations to be performed in an `unchecked` code block as the last statement within each `for` loop to optimize their execution cost.

Alleviation (2b9f60b6c3323fd234b570089ceff924cdb5851c):

The referenced loop iterator increment statements have been properly relocated to the end of each `for` loop's body and wrapped in an `unchecked` code block, optimizing their execution cost.

PHF-02C: Redundant Parenthesis Statement

Type	Severity	Location
Code Style	Informational	PauseHandlerFacet.sol:L89

Description:

The referenced statement is redundantly wrapped in parenthesis (`()`).

Example:

contracts/protocol/facets/PauseHandlerFacet.sol

SOL

```sol
89  if ((status.pauseScenario & (1 << i)) != 0) {
```

Recommendation:

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

Alleviation (2b9f60b6c3323fd234b570089ceff924cdb5851c):

The redundant parenthesis statement has been safely omitted.

ProtocolInitializationHandlerFacet Code Style Findings

PIH-01C: Inexistent Support for Sequential Upgrade

Type	Severity	Location
Code Style	Informational	ProtocolInitializationHandlerFacet.sol:L96, L98, L100

Description:

The `ProtocolInitializationHandlerFacet::initialize` function does not support upgrading a protocol from deployment directly to the latest version and instead executes the initialization of only one version.

Example:

contracts/protocol/facets/ProtocolInitializationHandlerFacet.sol

SOL

```
94  if (_isUpgrade) {
95      if (_version == bytes32("2.2.0")) {
96          initV2_2_0(_initializationData);
97      } else if (_version == bytes32("2.2.1")) {
98          initV2_2_1();
99      } else if (_version == bytes32("2.3.0")) {
100         initV2_3_0(_initializationData);
101     }
102 }
```

Recommendation:

We advise the code to support sequential initialization of versions by executing each version-suffixed initializer in sequence.

Alleviation (2b9f60b6c3323fd234b570089ceff924cdb5851c):

The Boson Protocol team evaluated this exhibit and has opted for a simpler approach instead of a sequential upgrade-path as the processing of the `_initializationData` would become more complex during sequential upgrades.

As such, we consider this exhibit as nullified given that we agree with the Boson Protocol team's assessment.

PIH-02C: Loop Iterator Optimization

Type	Severity	Location
Gas Optimization	Informational	ProtocolInitializationHandlerFacet.sol:L172

Description:

The linked `for` loop increments / decrements the iterator "safely" due to Solidity's built-in safe arithmetics (post-`0.8.X`).

Example:

contracts/protocol/facets/ProtocolInitializationHandlerFacet.sol

SOL

```sol
172 for (uint256 i = 0; i < sellerIds.length; i++) {
```

Recommendation:

We advise the increment / decrement operation to be performed in an `unchecked` code block as the last statement within the `for` loop to optimize its execution cost.

Alleviation (2b9f60b6c3323fd234b570089ceff924cdb5851c):

The referenced iterator is no longer present in the codebase, rendering this exhibit nullified.

SellerHandlerFacet Code Style Findings

SHF-01C: Loop Iterator Optimization

Type	Severity	Location
Gas Optimization	Informational	SellerHandlerFacet.sol:L277

Description:

The linked `for` loop increments / decrements the iterator "safely" due to Solidity's built-in safe arithmetics (post-`0.8.X`).

Example:

contracts/protocol/facets/SellerHandlerFacet.sol

SOL

```
277 for (i = 0; i < collectionCount; i++) {
```

Recommendation:

We advise the increment / decrement operation to be performed in an `unchecked` code block as the last statement within the `for` loop to optimize its execution cost.

Alleviation (2b9f60b6c3323fd234b570089ceff924cdb5851c):

The referenced iterator increment statement has been properly relocated to the end of the `for` loop body and wrapped in an `unchecked` code block, rendering this exhibit alleviated.

Finding Types

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

Input Sanitization

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

Indeterminate Code

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

Language Specific

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

Curve Specific

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

Code Style

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

Mathematical Operations

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

Logical Fault

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

Privacy Concern

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

Proof Concern

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

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.