OMNISCIA

August 6, 2024

SMART CONTRACT AUDIT REPORT

Fermion Protocol

Boson RWA Sale Verification Protocol

omniscia.io

info@omniscia.io

Online report: fermion-protocol-boson-rwa-sale-verification-protocol

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

Online report: fermion-protocol-boson-rwa-sale-verification-protocol

Boson RWA Sale Verification Protocol Security Audit

Audit Report Revisions

Commit Hash	Date	Audit Report Hash
4d07c7c662	July 12th 2024	0b63e5886d
876c9921ae	July 31st 2024	430d3bb740
43f5ee8c55	August 6th 2024	b5c4cda22e

Audit Overview

We were tasked with performing an audit of the Fermion Protocol codebase and in particular their RWA Sale Verification overlay system that integrates with the Boson Protocol.

The system is intricately complex and employs several concepts to aid in the protocol's implementation including a specialized entity ID system with granular per wallet roles, a custodial mechanism that requires constant upkeep through fees, a novel **EIP-20** and **EIP-721** contract implementation representing the fractionalized NFTs as well as their wrapped variants, and more.

The initial implementation incorporates the Boson Protocol price discovery mechanism using an OpenSea Seaport order to fulfil the sale of a RWA.

Over the course of the audit, we thoroughly analysed the codebase for any vulnerabilities that might arise from the code itself as well as throughout the various cross-contract interactions performed in the system.

The various implementations were matched with their in-line documentation where applicable as well as their description within the technical specification that the Fermion Protocol shared with us as part of the engagement.

We were able to uncover multiple complex vulnerabilities that arise from the multi-step processes involved in a Fermion Protocol exchange, such as a division truncation vulnerability that could result in an underflow operation during a different step of a RWA's verification with significant consequences.

We advise the Fermion 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 Fermion 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 Fermion Protocol and have identified that certain exhibits have not been adequately dealt with. We advise the Fermion Protocol team to revisit the following exhibits: `MTN-01M`, `MTN-02C`, `SWR-01M`, `FLB-01M`

Post-Audit Conclusion (43f5ee8c55)

The Fermion Protocol team evaluated our follow-up remediation chapters in the aforementioned list of exhibits and supplied us with a new commit hash that contains alleviation actions for them.

We have confirmed that all exhibits have been addressed properly either through acknowledgement or a proper remediative action.

As a result, we consider all outputs of the audit report properly consumed by the Fermion Protocol team and no pending actions to remain.

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	1	0	0	1
Informational	34	34	0	0
Minor	15	12	0	3
Medium	7	7	0	0
Major	5	5	0	0

During the audit, we filtered and validated a total of **5 findings utilizing static analysis** tools as well as identified a total of **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 can introduce potential misbehaviours of the system as well as exploits.

Scope

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

Target

- Repository: **https://github.com/fermionprotocol/contracts**
- Commit: 4d07c7c66204661ea9ca79512fbc704a5403b5e5
- Language: Solidity
- Network: Ethereum, Polygon
- Revisions: **4d07c7c662**, **876c9921ae**, **43f5ee8c55**

Contracts Assessed

File	Total Finding(s)
contracts/protocol/libs/Access.sol (ASS)	1
contracts/diamond/facets/AccessController.sol (ACR)	1
contracts/protocol/clients/Common.sol (CNO)	2
contracts/protocol/facets/Config.sol (CGI)	2
contracts/protocol/libs/Context.sol (CTX)	1
contracts/protocol/facets/Custody.sol (CYD)	0
contracts/protocol/domain/Constants.sol (CST)	0
contracts/protocol/libs/CustodyLib.sol (CLB)	2
contracts/protocol/facets/CustodyVault.sol (CVT)	2
contracts/diamond/Diamond.sol (DDN)	1

contracts/diamond/facets/DiamondCutFacet.sol (DCF)	0
contracts/diamond/facets/DiamondLoupeFacet.sol (DLF)	1
contracts/protocol/facets/Entity.sol (EYT)	4
contracts/protocol/domain/Errors.sol (ESR)	0
contracts/protocol/libs/EntityLib.sol (ELB)	2
contracts/protocol/facets/Funds.sol (FSD)	1
contracts/protocol/libs/FundsLib.sol (FLB)	1
contracts/protocol/clients/FermionFNFT.sol (FFN)	1
contracts/protocol/clients/FermionWrapper.sol (FWR)	3
contracts/protocol/clients/FermionFNFTBase.sol (FFF)	2
contracts/protocol/clients/FermionFractions.sol (FFS)	11
contracts/protocol/clients/FermionFractionsERC20Base.sol (FFE)	1
contracts/protocol/facets/Initialization.sol (INO)	2
contracts/diamond/libraries/LibDiamond.sol (LDD)	1

contracts/protocol/facets/MetaTransaction.sol (MTN)	6
contracts/protocol/facets/Offer.sol (ORE)	5
contracts/protocol/facets/Pause.sol (PES)	2
contracts/diamond/libraries/ReentrancyGuard.sol (RGD)	1
contracts/protocol/libs/Storage.sol (SEG)	2
contracts/protocol/clients/SeaportWrapper.sol (SWR)	2
contracts/protocol/domain/Types.sol (TSE)	0
contracts/protocol/facets/Verification.sol (VNO)	2

Compilation

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

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 `0.8.24` based on the version specified within the `hardhat.config.ts` 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` statements have been locked to `0.8.24` (`=0.8.24`), 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 **101 potential issues** within the codebase of which **93 were ruled out to be false positives** or negligible findings.

The remaining **8 issues** were validated and grouped and formalized into the **5 exhibits** that follow:

ID	Severity	Addressed	Title
FFF-01S	Informational	Yes	Inexistent Sanitization of Input Address
MTN-01S	Informational	Yes	Inexistent Sanitization of Input Address
ORE-01S	Informational	Yes	Inexistent Sanitization of Input Address
SWR-01S	Informational	Yes	Inexistent Sanitization of Input Addresses
VNO-01S	Informational	Yes	Inexistent Sanitization of Input Address

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in Fermion Protocol's Boson Protocol integrating RWA verification and fractionalization system.

As the project at hand implements a highly nuanced system involving multiple complex cross-contract interactions, 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 significant vulnerabilities** within the system which could have had **severe ramifications** to its overall operation; we advise all medium severity and above exhibits to be thoroughly evaluated and promptly addressed.

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 and the technical specification provided was immensely helpful in gaining a deeper understanding of the system.

A total of **57 findings** were identified over the course of the manual review of which **32 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	Informational	Yes	Inexistent Invocation of OpenZeppelin Initializer
CVT-01M	Unknown	Acknowledged	Unfair Dilution of Fractionalized NFTs
CVT-02M	Minor	Yes	Potentially Irrecoverable Auction State
DDN-01M	Major	Yes	Improper Application of Re-Entrancy Guard
EYT-01M	Minor	Yes	Inexistent Prevention of Self-Transfer

EYT-02M	Minor	Yes	Inexplicable Permittance of Multiple Entity Admins
EYT-03M	Major	Yes	Inexistent Validation of Recipient State
FFN-01M	Minor	Yes	Incompatibility of Approval System
FFF-01M	Medium	Yes	Inexistent Initialization of OpenZeppelin Dependency (EIP-721)
FFS-01M	Informational	Yes	Misleading Code Block
FFS-02M	Minor	Nullified	Incorrect Calculation of Remaining Votes Needed
FFS-03M	Minor	Acknowledged	Incorrect Minimum Bid Mechanism
FFS-04M	Minor	Acknowledged	Inexistent Automatic Extension of Auction
FFS-05M	Minor	Yes	Inexistent Restriction of Initial Bid

FFS-06M	Medium	Yes	EIP-721 Incompatibility
FFS-07M	Medium	Yes	Improper Handling of Maximum Fraction Bid
FFS-08M	Medium	Yes	Incorrect Calculation of Unlock Threshold
FFS-09M	Major	Yes	Incorrect Bounding of Bid Fractions
FFE-01M	Medium	Yes	Blockchain Explorer Incompatibility
FWR-01M	Informational	Yes	Inexplicable Exchange Token Configurations
FWR-02M	Minor	Yes	Incorrect Order of Operations
FLB-01M	Medium	Yes	Inexistent Validation of Token Type Deposited
LDD-01M	Informational	Yes	Unconditional Validation of Code Size

MTN-01M	Minor	Acknowledged	Inexistent Chain ID of Domain Type-Hash (EIP-712)
MTN-02M	Medium	Yes	Inconsistency of Hashed Payloads (EIP-1271)
ORE-01M	Minor	Yes	Double Accounting of Native Funds
ORE-02M	Minor	Yes	Inexistent Support of Native Token
ORE-03M	Minor	Yes	Seller-Defined Verification Time-Out
ORE-04M	Major	Nullified	Incorrect Calculation of Minimal Price w/ Boson Protocol Fee
RGD-01M	Major	Yes	Incorrect Re-Entrancy Guard Pattern
SWR-01M	Minor	Yes	Inexistent Restrictions of OpenSea Order (Fee)
VNO-01M	Minor	Yes	Inexistent Restriction of Verification Timeout

Code Style

During the manual portion of the audit, we identified **25 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
ASS-01C	Informational	Yes	Redundant Local Variable
CNO-01C	Informational	Nullified	Condition Short-Circuiting
CNO-02C	Informational	Nullified	Redundant Local Variable Declaration
CGI-01C	Informational	Yes	Generic Typographic Mistake
CGI-02C	Informational	Yes	Inefficient Initialization of Facet
CTX-01C	Informational	Yes	Ineffectual Usage of Safe Arithmetics
CLB-01C	Informational	Yes	Generic Typographic Mistake
CLB-02C	Informational	Yes	Tautological Assignment
DLF-01C	Informational	Yes	Redundant Variable Assignments
EYT-01C	Informational	Yes	Generic Typographic Mistake

ELB-01C	Informational	Yes	Non-Standard Invocation Style
ELB-02C	Informational	Yes	Redundant Local Variable Declaration
FFS-01C	Informational	Yes	Ineffectual Usage of Safe Arithmetics
FFS-02C	Informational	Nullified	Inefficient Conditional
FWR-01C	Informational	Yes	Redundant Re-Approval of Conduit
FSD-01C	Informational	Yes	Ineffectual Usage of Safe Arithmetics
INO-01C	Informational	Yes	Redundant Parenthesis Statement
INO-02C	Informational	Yes	Redundant Validation of Boson Protocol Address
MTN-01C	Informational	Yes	Potentially Improper Error
MTN-02C	Informational	Yes	Potentially Uncaught Error
MTN-03C	Informational	Yes	Redundant Function Implementation
PES-01C	Informational	Yes	Ineffectual Usage of Safe Arithmetics
PES-02C	Informational	Yes	Redundant Local Variable
SEG-01C	Informational	Yes	Inconsistent Storage Data Location Specifiers

SEG-02C	Informational	Yes	Inefficient Distinct Data Structures

FermionFNFTBase Static Analysis Findings

FFF-01S: Inexistent Sanitization of Input Address

Type	Severity	Location
Input Sanitization	Informational	FermionFNFTBase.sol:L21-L23

Description:

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

Impact:

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

Example:

contracts/protocol/clients/FermionFNFTBase.sol

SOL

```sol
21  constructor(address _bosonPriceDiscovery) {
22      BP_PRICE_DISCOVERY = _bosonPriceDiscovery;
23  }
```

Recommendation:

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

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

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

MetaTransaction Static Analysis Findings

MTN-01S: Inexistent Sanitization of Input Address

Type	Severity	Location
Input Sanitization	Informational	MetaTransaction.sol:L39-L44

Description:

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

Impact:

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

Example:

contracts/protocol/facets/MetaTransaction.sol

SOL

```sol
39  constructor(address _fermionProtocolAddress) {
40      FERMION_PROTOCOL_ADDRESS = _fermionProtocolAddress;
41      CHAIN_ID_CACHED = block.chainid;
42
43      DOMAIN_SEPARATOR_CACHED = buildDomainSeparator(PROTOCOL_NAME,
PROTOCOL_VERSION, FERMION_PROTOCOL_ADDRESS);
44  }
```

Recommendation:

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

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

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

Offer Static Analysis Findings

ORE-01S: Inexistent Sanitization of Input Address

Type	Severity	Location
Input Sanitization	Informational	Offer.sol:L35-L38

Description:

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

Impact:

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

Example:

contracts/protocol/facets/Offer.sol

SOL

```sol
35  constructor(address _bosonProtocol) {
36      BOSON_PROTOCOL = IBosonProtocol(_bosonProtocol);
37      BOSON_TOKEN = IBosonProtocol(_bosonProtocol).getTokenAddress();
38  }
```

Recommendation:

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

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

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

SeaportWrapper Static Analysis Findings

SWR-01S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Informational	SeaportWrapper.sol:L38-L47

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/clients/SeaportWrapper.sol

SOL

```sol
38  constructor(
39      address _bosonPriceDiscovery,
40      SeaportConfig memory _seaportConfig
41  ) FermionFNFTBase(_bosonPriceDiscovery) {
42      SEAPORT = _seaportConfig.seaport;
43      OS_CONDUIT = _seaportConfig.openSeaConduit == address(0)
44          ? _seaportConfig.seaport
45          : _seaportConfig.openSeaConduit;
46      OS_CONDUIT_KEY = _seaportConfig.openSeaConduitKey;
47  }
```

Recommendation:

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

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

Input sanitization was introduced for the `_seaportConfig.seaport` variable which represents the main address that should be sanitized within the function. As such, we consider this exhibit adequately addressed.

Verification Static Analysis Findings

VNO-01S: Inexistent Sanitization of Input Address

Type	Severity	Location
Input Sanitization	Informational	Verification.sol:L23-L25

Description:

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

Impact:

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

Example:

contracts/protocol/facets/Verification.sol

SOL

```sol
23  constructor(address _bosonProtocol) {
24      BOSON_PROTOCOL = IBosonProtocol(_bosonProtocol);
25  }
```

Recommendation:

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

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

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

AccessController Manual Review Findings

ACR-01M: Inexistent Invocation of OpenZeppelin Initializer

Type	Severity	Location
Standard Conformity	Informational	AccessController.sol:L26-L35

Description:

The `AccessController::initialize` function will not invoke the `AccessControlUpgradeable::__AccessControl_init` function which goes against the OpenZeppelin standard.

Impact:

While the OpenZeppelin `AccessControlUpgradeable` dependency does not presently require invocation due to being a no-op, it is still advisable to invoke these initialization functions in case their logic is updated.

Example:

contracts/diamond/facets/AccessController.sol

SOL

```
6   import { AccessControlUpgradeable as AccessControl } from "@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol";
```

Recommendation:

We advise the initializer to be properly invoked ensuring that the OpenZeppelin dependency has been properly configured post-deployment.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The `AccessController::__AccessControl_init` call has been introduced to the contract's initialization function addressing this exhibit.

CustodyVault Manual Review Findings

CVT-01M: Unfair Dilution of Fractionalized NFTs

Type	Severity	Location
Logical Fault	Unknown	CustodyVault.sol:L438, L456-L463

Description:

Whenever a forceful fractionalization occurs, all existing shares of other fractionalized NFTs will be unfairly diluted due to shares being minted without backing.

Impact:

A forceful fractionalization will lead to the dilution of existing shareholders which we consider an unfair trait of the system.

Example:

contracts/protocol/facets/CustodyVault.sol

SOL

```sol
435 // Forceful fractionalisation
436 if (itemsInVault > 0) {
437     // vault exist already
438     IFermionFNFT(fermionFNFTAddress).mintFractions(_tokenId, 1, 0);
439
440     CustodyLib.addItemToCustodianOfferVault(_tokenId, 1, 0, false, pl);
441 } else {
442     // no vault yet. Use the default parameters
443     FermionTypes.BuyoutAuctionParameters memory _buyoutAuctionParameters;
444     _buyoutAuctionParameters.exitPrice = pl.itemPrice[_tokenId];
```

Example (Cont.):

```sol
    uint256 partialAuctionThreshold = PARTIAL_THRESHOLD_MULTIPLIER * _custodianFee.amount;
    uint256 newFractionsPerAuction = (partialAuctionThreshold * DEFAULT_FRACTION_AMOUNT) /
        _buyoutAuctionParameters.exitPrice;
    FermionTypes.CustodianVaultParameters memory _custodianVaultParameters = FermionTypes
        .CustodianVaultParameters({
            partialAuctionThreshold: partialAuctionThreshold,
            partialAuctionDuration: _custodianFee.period / PARTIAL_AUCTION_DURATION_DIVISOR,
            liquidationThreshold: LIQUIDATION_THRESHOLD_MULTIPLIER * _custodianFee.amount,
            newFractionsPerAuction: newFractionsPerAuction
        });

    IFermionFNFT(fermionFNFTAddress).mintFractions(
        _tokenId,
        1,
        DEFAULT_FRACTION_AMOUNT,
        _buyoutAuctionParameters,
        _custodianVaultParameters,
        0
    );

    setupCustodianOfferVault(_tokenId, 1, _custodianVaultParameters, 0, false);
}
```

Recommendation:

We advise this approach to be revisited, as we consider it to be unfairly penalizing the original fractionalized shareholders which may be fully diligent with their NFT's upkeep.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The Fermion Protocol team evaluated this exhibit and while acknowledging the current design might be sub-optimal proceeded to retain it in place.

In detail, they envision that in the long-term fractionalized shareholders that were diluted would become whole (i.e. by the NFT becoming recombined) and consider that alternatives would be far too complex for the first iteration of the system.

As a result of these discussions, we consider this exhibit to be safely acknowledged and an item that the Fermion Protocol team intends to potentially address in a future iteration.

CVT-02M: Potentially Irrecoverable Auction State

Type	Severity	Location
Logical Fault	Minor	CustodyVault.sol:L346, L376

Description:

The `CustodyVault::bid` function permits any user to bid and will automatically create an entity ID for the bidder to store their refund in case they are outbid.

A problem arises from the way the bidding and auction finalization process works due to the fact that the system tracks the entity ID of the latest bidder rather than their address.

As an entity can be deleted via the `EntityFacet::deleteEntity` function, it is possible for an auction to be won by a bidder that cannot be processed via the `CustodyVault::endAuction` function.

Impact:

A bidder who either intentionally or accidentally deletes their entity ID will cause the auction to become irrecoverable as the `EntityLib::fetchEntityData` function call performed within `CustodyVault::endAuction` would fail.

To note, this might be a viable strategy to prevent further fractionalization increases as a new auction cannot be started once the previous one is on-going.

Example:

contracts/protocol/facets/CustodyVault.sol

SOL

```sol
367 function endAuction(uint256 _offerId) external
notPaused(FermionTypes.PausableRegion.CustodyVault) {
368     FermionStorage.ProtocolLookups storage pl = FermionStorage.protocolLookups();
369     FermionTypes.FractionAuction storage fractionAuction =
pl.fractionAuction[_offerId];
370
371     uint256 auctionEndTime = fractionAuction.endTime;
372     if (auctionEndTime == 0) revert AuctionNotStarted(_offerId);
373     if (auctionEndTime > block.timestamp) revert AuctionOngoing(_offerId,
fractionAuction.endTime);
374
375     // fractions to the winner
376     address winnerAddress =
EntityLib.fetchEntityData(fractionAuction.bidderId).admin;
```

Recommendation:

We advise the system to keep track of the latest bidder as an address, and to process refunds opportunistically if their bidder ID exists.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The Fermion Protocol team evaluated this exhibit and opted to omit the `EntityFacet::deleteEntity` function altogether, preventing created entities from being created and thus addressing this vulnerability indirectly.

Diamond Manual Review Findings

DDN-01M: Improper Application of Re-Entrancy Guard

Type	Severity	Location
Logical Fault	Major	Diamond.sol:L33

Description:

The `Diamond::fallback` function will apply the `ReentrancyGuard::nonReentrant` modifier regardless of whether a `staticcall` or `call` is being performed, resulting in a transaction failure for the former invocation types.

Impact:

Any protocol attempting to integrate with Fermion on-chain will fail to do when invoking read-only functions due to a re-entrancy guard that is applied in all call types.

Example:

contracts/diamond/Diamond.sol

SOL

```sol
// Find facet for function that is called and execute the
// function if a facet is found and return any value.
fallback() external payable nonReentrant {
    LibDiamond.DiamondStorage storage ds;
    bytes32 position = LibDiamond.DIAMOND_STORAGE_POSITION;
    // get diamond storage
    assembly {
        ds.slot := position
    }
    // get facet from function selector
```

Example (Cont.):

```sol
        address facet = ds.facetAddressAndSelectorPosition[msg.sig].facetAddress;
        if (facet == address(0)) {
            revert FunctionNotFound(msg.sig);
        }
        // Execute external function from facet using delegatecall and return any
value.
        assembly {
            // copy function selector and any arguments
            calldatacopy(0, 0, calldatasize())
            // execute function call using the facet
            let result := delegatecall(gas(), facet, 0, calldatasize(), 0, 0)
            // get any return value
            returndatacopy(0, 0, returndatasize())
            // return any return value or error back to the caller
            switch result
            case 0 {
                revert(0, returndatasize())
            }
            default {
                return(0, returndatasize())
            }
        }
    }
```

Recommendation:

As `TSTORE` operations are prohibited in `staticcall` (i.e. `view` / `pure`) calls, we advise setting the re-entrancy flag solely when a mutative call is involved.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The Fermion Protocol team opted to instead apply re-entrancy guards locally on each facet, ensuring that view-only calls do not impose these guards.

We validated that the re-entrancy guards have been adequately introduced across all state-mutating calls the Diamond will reference and thus consider this exhibit fully alleviated.

Entity Manual Review Findings

EYT-01M: Inexistent Prevention of Self-Transfer

Type	Severity	Location
Logical Fault	Minor	Entity.sol:L257-L271

Description:

The `EntityFacet::changeWallet` function can be presently invoked with a `_newWallet` argument of the caller themselves resulting in the wallet ID ownership being erased incorrectly.

Impact:

A user who attempts to re-assign themselves as the wallet owner will cause the wallet to become unowned instead.

Example:

contracts/protocol/facets/Entity.sol

SOL

```sol
257 function changeWallet(address _newWallet) external
notPaused(FermionTypes.PausableRegion.Entity) {
258     address msgSender = _msgSender();
259     FermionStorage.ProtocolLookups storage pl = FermionStorage.protocolLookups();
260     uint256 entityId = pl.entityId[msgSender];
261
262     if (entityId != 0) revert ChangeNotAllowed(); // to change the entity admin,
use setEntityAdmin and then revoke the old admin
263
264     uint256 walletId = pl.walletId[msgSender];
265     if (walletId == 0) revert NoSuchEntity(0);
266
```

Example (Cont.):

```sol
    pl.walletId[_newWallet] = walletId;
    delete pl.walletId[msgSender];

    emit WalletChanged(msgSender, _newWallet);
}
```

Recommendation:

We advise the code to either prevent a wallet change to oneself or to re-order the deletion and assignment statements so as to perform the assignment after the deletion.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The Fermion Protocol team opted to heed both of our two recommendations, re-ordering the deletion statement prior to the assignment and disallowing self-transfers altogether.

EYT-02M: Inexplicable Permittance of Multiple Entity Admins

Type	Severity	Location
Logical Fault	Minor	Entity.sol:L224-L225

Description:

The `EntityFacet::setEntityAdmin` function permits multiple administrators to be pending at once, however, pending administrators will overwrite the previous administrator of an entity.

As a result, a user who potentially relinquishes ownership of an entity to another may be able to re-capture ownership via a latent entity administrator approval.

Impact:

A user can set multiple pending administrators at once for an entity ID which is invalid as each pending administrator approval's consumption will result in the previous administrator being overwritten.

Example:

contracts/protocol/facets/Entity.sol

SOL

```sol
219 function setEntityAdmin(uint256 _entityId, address _wallet, bool _status)
external {
220     FermionStorage.ProtocolLookups storage pl = FermionStorage.protocolLookups();
221     EntityLib.validateEntityId(_entityId, pl);
222     validateEntityAdmin(_entityId, pl);
223
224     // set the pending admin
225     pl.pendingEntityAdmin[_entityId][_wallet] = _status;
226
227     EntityLib.storeCompactWalletRole(
228         _entityId,
```

Example (Cont.):

```sol
        _wallet,
        0xff << (31 * BYTE_SIZE),
        _status,
        pl,
        FermionStorage.protocolEntities()
    ); // compact role for all current and potential future roles
    EntityLib.emitAdminWalletAddedOrRemoved(_entityId, _wallet, _status);
}
```

Recommendation:

We advise the system to store a single pending entity administrator per entity ID preventing the scenario outlined.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The system was updated to keep track of and permit a single pending entity administrator at a time, preventing the misbehaviors outlined.

EYT-03M: Inexistent Validation of Recipient State

Type	Severity	Location
Logical Fault	Major	Entity.sol:L257-L271

Description:

The `EntityFacet::changeWallet` function can be utilized to overwrite the `walletId` entry of another user without their consent thereby causing them to potentially lose wallet-level access to their entity.

Impact:

Given that wallet creations are free, a user can utilize this mechanism to sabotage access to a particular wallet and thereby prevent withdrawals from being executable.

Example:

contracts/protocol/facets/Entity.sol

SOL

```sol
257 function changeWallet(address _newWallet) external
notPaused(FermionTypes.PausableRegion.Entity) {
258     address msgSender = _msgSender();
259     FermionStorage.ProtocolLookups storage pl = FermionStorage.protocolLookups();
260     uint256 entityId = pl.entityId[msgSender];
261
262     if (entityId != 0) revert ChangeNotAllowed(); // to change the entity admin,
use setEntityAdmin and then revoke the old admin
263
264     uint256 walletId = pl.walletId[msgSender];
265     if (walletId == 0) revert NoSuchEntity(0);
266
```

Example (Cont.):

```sol
    pl.walletId[_newWallet] = walletId;
    delete pl.walletId[msgSender];

    emit WalletChanged(msgSender, _newWallet);
}
```

Recommendation:

We advise the code to ensure that the recipient `_newWallet` wallet ID is `0` preventing an unauthorized overwrite from being executable.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The system was updated to prevent transfers to `_newWallet` addresses that already possess a wallet ID, ensuring that wallet overwrites cannot occur via the **`EntityFacet::changeWallet`** function.

FermionFNFT Manual Review Findings

FFN-01M: Incompatibility of Approval System

Type	Severity	Location
Standard Conformity	Minor	FermionFNFT.sol:L142-L151

Description:

Most **EIP-20** integrating protocols will employ a `FermionFNFT::approve` operation with the maximum possible `uint256` (`type(uint256).max`) which is incompatible with the `FermionFNFT` dual EIP token system.

Impact:

Most readily-available DeFi protocols that employ maximum approvals are incompatible with the fractionalized **EIP-20** token of the Fermion Protocol.

Example:

contracts/protocol/clients/FermionFNFT.sol

SOL

```sol
142 function approve(address to, uint256 tokenIdOrBalance) public virtual
override(IERC721, ERC721) {
143     if (tokenIdOrBalance > type(uint128).max) {
144         ERC721.approve(to, tokenIdOrBalance);
145     } else {
146         bool success = approveFractions(to, tokenIdOrBalance);
147         assembly {
148             return(success, 32)
149         }
150     }
151 }
```

Recommendation:

We advise this incompatibility as well as the incompatibility of the `FermionFNFT::transferFrom` and `FermionFNFT::balanceOf` functions to be re-assessed as they can result in complications when the Fermion Protocol is integrated by third parties.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The code of the system was adjusted to handle invocations of `type(uint256).max` as maximum approvals, ensuring at least a base level of support with external protocols.

To note, the discrepancies in the function's behavior can still result in vulnerabilities within third-party systems and all integrations **must be made aware of the full discrepancy list of the Fermion Protocol fractionalized dual EIP system**.

We consider this exhibit addressed based on the fact that approvals are adequately handled and that the Fermion Protocol team will update their documentation to ensure integrators are fully aware of the contract's unique caveats.

FermionFNFTBase Manual Review Findings

FFF-01M: Inexistent Initialization of OpenZeppelin Dependency (EIP-721)

Type	Severity	Location
Standard Conformity	Medium	FermionFNFTBase.sol:L11

Description:

The `ERC721Upgradeable` dependency of OpenZeppelin remains uninitialized across all children of the `FermionFNFTBase` contract resulting in the `IERC721::name` and `IERC721::symbol` functions of the system to yield an empty string.

Impact:

An **EIP-721** asset that does not expose a `name` and `symbol` is frequently labelled as an unwanted asset in off-chain software that might significantly hinder the adoption of the Fermion fractionalized NFTs, especially in the case of secondary price discovery markets such as OpenSea's Seaport.

Example:

contracts/protocol/clients/FermionFNFTBase.sol

SOL

```sol
4   import { ERC721Upgradeable as ERC721 } from "@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol";
5
6   /**
7    * @title FermionFNFTBase
8    * @notice Base erc721 upgradeable contract for Fermion FNFTs
9    *
10   */
11  contract FermionFNFTBase is ERC721 {
12      // Contract addresses
13      address internal fermionProtocol;
```

Example (Cont.):

```sol
    address internal voucherAddress;
    address internal immutable BP_PRICE_DISCOVERY; // Boson protocol Price Discovery client

    /**
     * @notice Constructor
     *
     */
    constructor(address _bosonPriceDiscovery) {
        BP_PRICE_DISCOVERY = _bosonPriceDiscovery;
    }
}
```

Recommendation:

We advise the `ERC721Upgradeable` dependency to be initialized either via a utility function exposed via the `FermionFNFTBase` contract or via a new declaration in one of its children, ensuring that the metadata the **EIP-721** asset yields are correct.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The dependency is now initialized at the `FermionFNFT::initialize` function, ensuring a proper name and abbreviation is configured for the **EIP-721** asset.

FermionFractions Manual Review Findings

FFS-01M: Misleading Code Block

Type	Severity	Location
Logical Fault	Informational	FermionFractions.sol:L799-L802

Description:

The referenced code block will erase the auction parameters of the system, however, its comment is slightly misleading as it states that such an action will:

> allow fractionalisation with new parameters

This statement is not correct as fractionalization with new parameters can be performed **the moment an auction starts** as the `nftCount` is decremented at the start of the auction rather than the end of it.

Example:

contracts/protocol/clients/FermionFractions.sol

SOL

```sol
799 if ($.nftCount == 0) {
800     // allow fractionalisation with new parameters
801     delete $.auctionParameters;
802 }
```

Recommendation:

In light of this discrepancy, we advise the system to either disallow a fractionalization when non-zero auction parameters have been configured or to omit the referenced `if` block as it is ineffective.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The referenced code block was omitted per the latter of our two recommendations, indicating that it is not a requirement for the configuration to be erasable solely at that point.

FFS-02M: Incorrect Calculation of Remaining Votes Needed

Type	Severity	Location
Logical Fault	Minor	FermionFractions.sol:L240, L247

Description:

The `FermionFractions::voteToStartAuction` function will permit a caller to deposit more fractions than needed to start the auction due to an incorrect `availableFractions` calculation.

In detail, the function will cap the votes to the total `fractionsPerToken` even though the votes needed for an auction to start are a percentage of it.

Impact:

A user who wants a vote to start and attempts to cast the "maximum" will cause more fractions than needed to be consumed and locked from their balance for the duration of the auction.

Example:

contracts/protocol/clients/FermionFractions.sol

SOL

```
235 uint256 fractionsPerToken = liquidSupply() / $.nftCount;
236
237 FermionTypes.Votes storage votes = auction.votes;
238 uint256 availableFractions = fractionsPerToken - votes.total;
239
240 if (_fractionAmount > availableFractions) _fractionAmount = availableFractions;
241
242 _transferFractions(msgSender, address(this), _fractionAmount);
243
244 votes.individual[msgSender] += _fractionAmount;
```

Example (Cont.):

```sol
votes.total += _fractionAmount;

if (votes.total > (fractionsPerToken * $.auctionParameters.unlockThreshold) / HUNDRED_PERCENT) {
    startAuction(_tokenId);
}

emit Voted(_tokenId, msgSender, _fractionAmount);
```

Recommendation:

We advise the code to cap the `availableFractions` based on the unlock threshold rather than the total fractions needed to acquire the NFT, ensuring that a casted vote will consume solely the fractions needed.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The Fermion Protocol team evaluated this exhibit and stated that while they consider the use case of a user being able to supply more fractions than needed desirable, the `availableFractions` calculation was not taking into account the already `lockedFractions`.

They then proceeded to ensure the `lockedFractions` are accounted for, preventing the `availableFractions` from being higher than permitted in total.

We consider the original exhibit to be nullified as it describes desirable behavior by the Fermion Protocol team.

FFS-03M: Incorrect Minimum Bid Mechanism

Type	Severity	Location
Logical Fault	Minor	FermionFractions.sol:L311-L314

Description:

The minimum bid mechanism that also functions as the outbidding conditional is presently incorrect as it solely factors in the `_price` that the user supplies rather than taking into account the `_fractions` as well.

This represents a flaw in the overall fiscal outcome of the auction and permits a user to outbid another with the same "price" once the ideal price of an asset has been achieved in the auction.

In detail, let us consider an asset that is fairly priced at `1_000` USDC. This bid was made by supplying a `_price` of `1_000`.

Given that this price represents the fair value of the asset, a user can go on the open market and purchase `500` USDC worth of shares for the `FermionFNFT`. The same user can proceed with placing a bid of `2_000` USDC for the `_price` and 50% of the `_fractions` necessary to unlock the NFT.

This will result in the outbidding process to pass successfully even though the user did not expend more funds than the previous bidder.

Impact:

The overall auction process can be considered unfair in its current implementation as a user can outbid another without expending more funds than the original one.

Example:

contracts/protocol/clients/FermionFractions.sol

SOL

```sol
304 function bid(uint256 _tokenId, uint256 _price, uint256 _fractions) external
payable {
305     FermionTypes.BuyoutAuctionStorage storage $ = _getBuyoutAuctionStorage();
306     FermionTypes.Auction storage auction = getLastAuction(_tokenId, $);
307     FermionTypes.AuctionDetails storage auctionDetails = auction.details;
308
309     if (!$.isFractionalised[_tokenId]) revert TokenNotFractionalised(_tokenId);
310
311     uint256 minimalBid = (auctionDetails.maxBid * (HUNDRED_PERCENT +
MINIMAL_BID_INCREMENT)) / HUNDRED_PERCENT;
312     if (_price < minimalBid) {
313         revert InvalidBid(_tokenId, _price, minimalBid);
```

Example (Cont.):

```sol
    }
```

Recommendation:

We advise the minimum bid / outbid mechanism to incorporate the `_fractions` that are employed to minimize this attack vector.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

This exhibit was extensively reviewed by the Fermion Protocol team and we deliberated its validity, viability in a production environment, and overall impact throughout our discussions with them.

We concluded that an indirect advantage for fractional owner bidders will always be present due to the current system's design, and that it is an acceptable risk for this iteration of the system.

FFS-04M: Inexistent Automatic Extension of Auction

Type	Severity	Location
Logical Fault	Minor	FermionFractions.sol:L319-L323

Description:

The `AUCTION_END_BUFFER` extension system that is employed in the `CustodyVault` has not been incorporated into the auctions of the `FermionFractions` contract even though the `AUCTION_END_BUFFER` constant is imported.

Impact:

An auction will not be automatically extended if a bid occurs towards its end in contrast to the `CustodyVault` auction implementation.

Example:

contracts/protocol/clients/FermionFractions.sol

SOL

```
319 if (auctionDetails.state >= FermionTypes.AuctionState.Ongoing) {
320     if (block.timestamp > auctionDetails.timer) revert AuctionEnded(_tokenId,
auctionDetails.timer);
321
322     fractionsPerToken = auctionDetails.totalFractions;
323 } else {
```

Recommendation:

We advise the automatic extension system to be properly implemented, preventing last-minute gas-based bidding wars from being waged.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The import of the `AUCTION_END_BUFFER` in the `FermionFractions` contract has been removed, indicating that the buffer is not meant to be in effect within it.

As such, we consider this exhibit adequately acknowledged as the Fermion Protocol team wishes to not impose this automatic extension mechanism as advised.

FFS-05M: Inexistent Restriction of Initial Bid

Type	Severity	Location
Mathematical Operations	Minor	FermionFractions.sol:L311

Description:

While the Fermion Protocol documentation states that auctions should start at a value of `0`, a misbehaviour will arise if the price of the auction remains `0` throughout its duration.

Specifically, the minimum bid increment will be `0` as a percentage of `0` resulting in an outbid being possible via the exact same price.

Impact:

For undesirable assets that might remain at a price of `0` and be solely bid on with fractions, an outbid will be possible with the exact same price configuration.

Example:

contracts/protocol/clients/FermionFractions.sol

SOL

```sol
311 uint256 minimalBid = (auctionDetails.maxBid * (HUNDRED_PERCENT +
MINIMAL_BID_INCREMENT)) / HUNDRED_PERCENT;
312 if (_price < minimalBid) {
313     revert InvalidBid(_tokenId, _price, minimalBid);
314 }
```

Recommendation:

We advise the `if` conditional that evaluates the `minimalBid` to ensure the `_price` is strictly greater than the minimal bid, thereby guaranteeing that an increase always occurs even if a rounding operation has caused the `minimalBid` to be the same as the `_price`.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The minimal bid calculation mechanism was updated to ensure that, if it matches the current maximum bid, the new bid is at least greater by `1` to prevent equal-value bids from outbidding one another.

FFS-06M: EIP-721 Incompatibility

Type	Severity	Location
Standard Conformity	Medium	FermionFractions.sol:L534-L538

Description:

The `FermionFractions::balanceOf` implementation will satisfy the **EIP-20** interface rendering the contract incompatible with **EIP-721** off-chain and on-chain systems.

In turn, this might result in misbehaviours when the `FermionFNFT` is integrated by third-party implementations.

Impact:

The compatibility of the top-level `FermionFNFT` implementation with off-chain software is questionable and will cause it to be considered a potentially incompatible asset in blockchain explorers, third-party marketplaces, and other highly sensitive applications.

Example:

contracts/protocol/clients/FermionFractions.sol

SOL

```sol
529 /**
530  * @notice Returns the number of fractions. Represents the ERC20 balanceOf method
531  *
532  * @param _owner The address to check
533  */
534 function balanceOf(
535     address _owner
536 ) public view virtual override(ERC721, FermionFractionsERC20Base) returns (uint256)
{
537     return FermionFractionsERC20Base.balanceOf(_owner);
538 }
```

Recommendation:

As the `event` related exhibit outlines, we strongly advise either **EIP-20** or **EIP-721** to be selected as the canonical EIP the `FermionFNFT` incorporates and to implement the other EIP's related functions in a non-compliant way, ensuring full compatibility of the `FermionFNFT` asset when treated as either **EIP-721** or **EIP-20**.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The Fermion Protocol team opted to retain **EIP-721** compatibility in the main "token" contract of the system and thus updated the `FermionFNFT` contract to implement a **`FermionFNFT::balanceOf`** function that exposes the **EIP-721** balance of the user.

FFS-07M: Improper Handling of Maximum Fraction Bid

Type	Severity	Location
Logical Fault	Medium	FermionFractions.sol:L349

Description:

The `FermionFractions::bid` process will permit users to bid with a mixture of fractions and tokens which can be entirely one-sided.

In case a user supplies a bid that is fully represented by fractions, the `FermionFractions::bid` function will not sanitize the input `_price` and will use whatever the user assigned. Given that the outbidding process evaluates the `_price` and not the shares of a user, it is possible for a full-share bid to be outbid by another with a higher `_price` that does not affect how the bid is ultimately processed.

Impact:

A user with a full-fraction bid may be outbid by another without any actual increase in the fiscal outcome of the auction.

Example:

contracts/protocol/clients/FermionFractions.sol

SOL

```sol
346 uint256 bidAmount;
347 if (bidderFractions > fractionsPerToken) {
348     // bidder has enough fractions to claim a full NFT without paying anything.
Does a price matter in this case?
349     bidderFractions = fractionsPerToken;
350 } else {
351     bidAmount = ((fractionsPerToken - bidderFractions) * _price) /
fractionsPerToken;
352 }
```

Recommendation:

We advise the code to assign the maximum `_price` possible, preventing a full-fraction bid from being outbid as that should effectively signal the end of the auction.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The system was refactored and introduced a new auction state called `Reserved` that is triggered whenever a user makes a bid that permits them to claim the NFT solely with fractions.

Such an operation will prevent further bidding to occur, effectively finalizing the auction and causing the vulnerability described to never be possible.

FFS-08M: Incorrect Calculation of Unlock Threshold

Type	Severity	Location
Mathematical Operations	Medium	FermionFractions.sol:L90-L92, L247

Description:

The unlock threshold calculation in the `FermionFractions::voteToStartAuction` function is non-inclusive thereby leading to a vulnerability when it is configured as a `HUNDRED_PERCENT`.

In detail, the auction will never be able to start via a vote if the configured unlock threshold is `100%` as the maximum possible votes are capped to `fractionsPerToken - votes.total`, and the unlock threshold will always be `fractionsPerToken` which is matched in a non-inclusive way.

Impact:

An unlock threshold of 100% that is accepted by the system will cause the auction to never be able to start via voting.

Example:

contracts/protocol/clients/FermionFractions.sol

SOL

```sol
235 uint256 fractionsPerToken = liquidSupply() / $.nftCount;
236
237 FermionTypes.Votes storage votes = auction.votes;
238 uint256 availableFractions = fractionsPerToken - votes.total;
239
240 if (_fractionAmount > availableFractions) _fractionAmount = availableFractions;
241
242 _transferFractions(msgSender, address(this), _fractionAmount);
243
244 votes.individual[msgSender] += _fractionAmount;
```

Example (Cont.):

```sol
votes.total += _fractionAmount;

if (votes.total > (fractionsPerToken * $.auctionParameters.unlockThreshold) / HUNDRED_PERCENT) {
    startAuction(_tokenId);
}
```

Recommendation:

We advise the comparison to become inclusive, ensuring that a configuration of a 100% can be properly processed by the system.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The unlock threshold comparison has been updated to be inclusive, ensuring that the percentage configured can always be met.

FFS-09M: Incorrect Bounding of Bid Fractions

Type	Severity	Location
Mathematical Operations	Major	FermionFractions.sol:L347-L350, L792

Description:

The current upper bound of a bidder's fractions is incorrect and can result in an auction that can never be finalized as it does not take into account the fractions that have already been "expended" for the NFT by the voters.

In detail, the `FermionFractions::finalizeAuction` function will calculate the unrestricted redeemable supply's increase as `fractionsPerToken - lockedVotes - winnersLockedFractions`.

When any non-zero votes are assigned to an auction and the `winnersLockedFractions` match 100% (i.e. the winner was not a voter), the calculation will result in an uncaught underflow thereby denying the auction's fulfilment permanently.

Impact:

An auction that has had any vote cast to it has a moderate likelihood of becoming impossible to finalize, thereby affecting a significant feature of the system permanently and directly affecting funds and assets.

Example:

contracts/protocol/clients/FermionFractions.sol

SOL

```
792 $.unrestricedRedeemableSupply += (fractionsPerToken - lockedVotes -
winnersLockedFractions);
```

Recommendation:

We advise the bidding process to permit up to `fractionsPerToken - votes.total` fractions to be supplied ensuring the cumulative fractions that have been "reserved" for an auction cannot exceed the total fractions needed to immediately unlock the NFT.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The bidding mechanism was updated to ensure that a prospective bidder can provide up to `fractionsPerToken - votes.total` towards the auction thereby preventing the uncaught underflow from ever manifesting.

FermionFractionsERC20Base Manual Review Findings

FFE-01M: Blockchain Explorer Incompatibility

Type	Severity	Location
Standard Conformity	Medium	FermionFractionsERC20Base.sol:L188

Description:

The `FermionFractionsERC20Base` **EIP-20** implementation will emit the same event signature as an **EIP-721** transfer which will cause contradictory information to be captured by blockchain explorers.

Impact:

The compatibility of the top-level `FermionFNFT` implementation with off-chain software is questionable and will cause it to be considered a potentially incompatible asset in blockchain explorers, third-party marketplaces, and other highly sensitive applications.

Example:

contracts/protocol/clients/FermionFractionsERC20Base.sol

SOL

```
188 emit IERC721.Transfer(from, to, value); // ERC20 Transfer event and ERC721 Transfer events have the same signature
```

Recommendation:

We strongly advise the top-level `FermionFNFT` implementation to adhere to a single token implementation canonically (either **EIP-20** or **EIP-721**), and the secondary token implementation to be supported in a non-compliant way.

The present system is non-compliant with either standard causing it to be poorly processed and potentially blacklisted by off-chain software, including integral software the protocol intends to interface with such as OpenSea's Seaport.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The Fermion Protocol team evaluated the **EIP-721** and **EIP-20** inconsistencies present in the singleton token contract and opted to support **EIP-721** in full with **EIP-20** support being incomplete.

As a result, the referenced event was replaced by a custom `FractionsTransfer` event thereby no longer leading to inconsistencies in off-chain explorers.

FermionWrapper Manual Review Findings

FWR-01M: Inexplicable Exchange Token Configurations

Type	Severity	Location
Logical Fault	Informational	FermionWrapper.sol:L77, L92

Description:

The referenced assignments of the `_exchangeToken` yielded by the unwrapping operations are inexplicable as a different exchange token notion is employed throughout the auction system.

Example:

contracts/protocol/clients/FermionWrapper.sol

SOL

```sol
60  /**
61   * @notice Unwraps the voucher, finalizes the auction, transfers the Boson rNFT to Fermion Protocol and F-NFT to the buyer
62   *
63   * @param _tokenId The token id.
64   * @param _buyerOrder The Seaport buyer order.
65   */
66  function unwrap(uint256 _tokenId, SeaportTypes.AdvancedOrder calldata _buyerOrder) external {
67      unwrap(_tokenId);
68
69      (, address _exchangeToken) = finalizeAuction(_tokenId, _buyerOrder);
```

Example (Cont.):

```sol

        // Transfer token to protocol
        // N.B. currently price is always 0. This is a placeholder for future use, when other PD mechanisms will be supported
        // if (price > 0) {
        //     IERC20(_exchangeToken).safeTransfer(BP_PRICE_DISCOVERY, price);
        // }

        Common._getFermionCommonStorage().exchangeToken = _exchangeToken;
    }

    /**
     * @notice Unwraps the voucher, but skip the OS auction and leave the F-NFT with the seller
     *
     * @param _tokenId The token id.
     */
    function unwrapToSelf(uint256 _tokenId, address _exchangeToken, uint256 _verifierFee) external {
        unwrap(_tokenId);

        if (_verifierFee > 0) {
            IERC20(_exchangeToken).safeTransfer(BP_PRICE_DISCOVERY, _verifierFee);
        }

        Common._getFermionCommonStorage().exchangeToken = _exchangeToken;
    }
```

Recommendation:

We advise the referenced assignments and variable to be omitted as they are not utilized anywhere else.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

Both of the referenced assignments have been omitted, optimizing the code's legibility and structure.

FWR-02M: Incorrect Order of Operations

Type	Severity	Location
Logical Fault	Minor	FermionWrapper.sol:L151-L152

Description:

The `FermionWrapper::wrap` function will mint the wrapped token ID to the `_to` recipient in a re-entrant way prior to configuring the token's state to `Wrapped`, thereby permitting it to be freely transferred during the **EIP-721** callback incorrectly.

Impact:

An invariant of the system is breached during a re-entrant call when wrapping whereby a token ID that is `Wrapped` can be freely transferred.

Example:

contracts/protocol/clients/FermionWrapper.sol

SOL

```sol
142 function wrap(uint256 _firstTokenId, uint256 _length, address _to) internal {
143     for (uint256 i = 0; i < _length; i++) {
144         uint256 tokenId = _firstTokenId + i;
145 
146         // Transfer vouchers to this contract
147         // Not using safeTransferFrom since this contract is the recipient and we
are sure it can handle the vouchers
148         IERC721(voucherAddress).transferFrom(_msgSender(), address(this),
tokenId);
149 
150         // Mint to the specified address
151         _safeMint(_to, tokenId);
```

Example (Cont.):

```sol
        Common.changeTokenState(tokenId, FermionTypes.TokenState.Wrapped);
    }
    wrapOpenSea();
}
```

Recommendation:

We advise the code to either invoke the non-reentrant **EIP-721** mint function, or to mint the token, set its status, and then invoke the **EIP-721** callback to the intended recipient.

Either of the two approaches are considered valid and will adhere to the Checks-Effects-Interactions pattern.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The code of the `FermionFNFT` contract was updated to perform the wrapping operation instead during its `_update` operation, ensuring that the token's state has been configured prior to the recipient re-entrant call and rendering this exhibit alleviated.

FundsLib Manual Review Findings

FLB-01M: Inexistent Validation of Token Type Deposited

Type	Severity	Location
Logical Fault	Medium	FundsLib.sol:L64

Description:

The `FundsFacet` and `FundsLib` contract implementations do not impose any restrictions on the `_tokenAddress` deposited, permitting an **EIP-721** asset to be deposited as an **EIP-20** asset.

The current implementation of the `FermionFNFT` results in a potentially significant vulnerability arising from the `Funds` and `FundsLib` implementation whereby an NFT is deposited but fractions are withdrawn, permitting the user to withdraw more fractions than their NFT would be worth.

Impact:

While **EIP-721** may be presently deposited to the system, they will become irrecoverable as the **`FundsLib::transferFundsFromProtocol`** function will execute an `IERC20::transfer` that is incompatible with the **EIP-721** standard.

However, the specialized `FermionFNFT` implementation permits a malicious user to exploit this vulnerability by depositing an NFT which will provide a significantly high allowance to an entity, and to withdraw less funds than they deposited thereby withdrawing shares instead of NFTs.

This will permit a user to withdraw an arbitrarily higher number of fractions in **EIP-20** form than their **EIP-721** asset would be worth.

Example:

contracts/protocol/libs/FundsLib.sol

SOL

```sol
47  /**
48   * @notice Tries to transfer tokens from the caller to the protocol.
49   *
50   * Reverts if:
51   * - Contract at token address does not support ERC20 function transferFrom
52   * - Calling transferFrom on token fails for some reason (e.g. protocol is not
approved to transfer)
53   * - Received ERC20 token amount differs from the expected value
54   *
55   * @param _tokenAddress - address of the token to be transferred
56   * @param _from - address to transfer funds from
```

Example (Cont.):

```sol
  * @param _amount - amount to be transferred
  */
 function transferFundsToProtocol(address _tokenAddress, address _from, uint256 _amount) internal {
     // protocol balance before the transfer
     uint256 protocolTokenBalanceBefore = IERC20(_tokenAddress).balanceOf(address(this));

     // transfer ERC20 tokens from the caller
     IERC20(_tokenAddress).safeTransferFrom(_from, address(this), _amount);

     // protocol balance after the transfer
     uint256 protocolTokenBalanceAfter = IERC20(_tokenAddress).balanceOf(address(this));

     // make sure that expected amount of tokens was transferred
     uint256 receivedAmount = protocolTokenBalanceAfter - protocolTokenBalanceBefore;
     if (receivedAmount != _amount) revert FundsErrors.WrongValueReceived(_amount, receivedAmount);
 }
```

Recommendation:

We advise the code to ensure that the asset attempted to be deposited is not an **EIP-721** one by performing an `IERC721::getApproved` call and reverting if it succeeds.

Alleviation (876c9921ae):

A `try-catch` clause was introduced at the beginning of the **`FundsLib::transferFundsToProtocol`** function that evaluates whether the asset deposited is an **EIP-721** asset through the **EIP-165**.

While the adjustment adequately protects the contract against **EIP-721** assets, it may result in an uncaught decoding error similarly to **`MTN-02C`** and the check should be handled accordingly to ensure as wide of a compatibility as posible with **EIP-20** assets.

Alleviation (43f5ee8c55):

The same follow-up alleviation approach as **`MTN-02C`** was applied here, invoking the **EIP-165** interface support function using a low-level `staticcall` on the `_tokenAddress` and thus permitting the yielded payload to be decoded gracefully.

As such, we consider this submission fully alleviated.

LibDiamond Manual Review Findings

LDD-01M: Unconditional Validation of Code Size

Type	Severity	Location
Logical Fault	Informational	LibDiamond.sol:L158

Description:

The `LibDiamond::initializeDiamondCut` function will unconditionally enforce the `_init` address has contract code thereby preventing usage of a self-target during deployment.

Impact:

Although the described pattern is not applied in the current iteration of the codebase, it is still advisable to support it in case of a future deployment of the system under a new format.

Example:

contracts/diamond/libraries/LibDiamond.sol

SOL

```sol
154 function initializeDiamondCut(address _init, bytes memory _calldata) internal {
155     if (_init == address(0)) {
156         return;
157     }
158     enforceHasContractCode(_init, "LibDiamondCut: _init address has no code");
```

Recommendation:

We advise the code to be reverted to its original state as represented in the **EIP-2535** standard, enforcing contract code existence solely when the `_init` address is not equal to the contract executing the logic itself (i.e. `address(this)`).

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The referenced contract code check has been wrapped in an `if` block as advised, ensuring that the system adheres to the reference **EIP-2535** implementation.

MetaTransaction Manual Review Findings

MTN-01M: Inexistent Chain ID of Domain Type-Hash (EIP-712)

Type	Severity	Location
Standard Conformity	Minor	MetaTransaction.sol:L20-L21, L295-L310

Description:

The referenced `EIP712_DOMAIN_TYPEHASH` does not include the `block.chainid` as an argument and instead employs the `bytes32 salt` argument with the chain ID value.

Per the **EIP-712** standard itself, a `bytes32 salt` value should indicate protocol differentiation and be utilized as a last resort whereas the `uint256 chainId` variable (EIP-155) is better integrated by off-chain software and will achieve better validation of the contract's domain.

Impact:

Off-chain software might be incapable of properly integrating with the **EIP-712** domain type-hash as defined in the `MetaTransactionFacet` due to its non-standard argument declarations and construction.

Example:

contracts/protocol/facets/MetaTransaction.sol

SOL

```sol
20  bytes32 private constant EIP712_DOMAIN_TYPEHASH =
21      keccak256(bytes("EIP712Domain(string name,string version,address
verifyingContract,bytes32 salt)"));
```

Recommendation:

We advise the `EIP712_DOMAIN_TYPEHASH` to be updated incorporating the canonical `uint256 chainId` argument thereby increasing the off-chain software compatibility of the contract.

Alleviation (43f5ee8c554ed321fef52435635b172f36a5182c):

The Fermion Protocol team evaluated this submission and clarified that they intend to retain the `chainId` as a component of the `salt` rather than the **EIP-712** typehash itself to permit a user's wallet connected to one network to be able to generate signed payloads for others without switching networks.

Given that cross-chain attacks are still prevented via the `salt` parameter, we consider this approach to be secure albeit non-standard and thus consider the exhibit to be acknowledged.

MTN-02M: Inconsistency of Hashed Payloads (EIP-1271)

Type	Severity	Location
Standard Conformity	Medium	MetaTransaction.sol:L334, L351

Description:

The `MetaTransaction::verify` function is meant to support **EIP-1271** and ECDSA based signature validation simultaneously, however, it contradicts the **EIP-1271** standard by forwarding a different hashed payload to the `IERC1271` contract and validating a different hashed payload via the `ecrecover` instruction.

Impact:

The inconsistency of the hashed payloads might cause properly compliant **EIP-1271** contracts to fail when integrated with the `MetaTransaction` facet.

Example:

contracts/protocol/facets/MetaTransaction.sol

SOL

```sol
325 function verify(
326     address _user,
327     bytes32 _hashedMetaTx,
328     bytes32 _sigR,
329     bytes32 _sigS,
330     uint8 _sigV
331 ) internal view returns (bool) {
332     // Check if user is a contract implementing ERC1271
333     if (_user.code.length > 0) {
334         try IERC1271(_user).isValidSignature(_hashedMetaTx, abi.encodePacked(_sigR, _sigS, _sigV)) returns (
```

Example (Cont.):

```sol
            bytes4 magicValue
        ) {
            if (magicValue != IERC1271.isValidSignature.selector) revert InvalidSignature();
            return true;
        } catch {
            revert InvalidSignature();
        }
    }

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

    address signer = ecrecover(toTypedMessageHash(_hashedMetaTx), _sigV, _sigR, _sigS);
    if (signer == address(0)) revert InvalidSignature();
    return signer == _user;
}
```

Recommendation:

We advise the same payload to be utilized across validation schemes in accordance to the EIP itself and its canonical implementations (i.e. OpenZeppelin).

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The `MetaTransaction::verify` function was updated to calculate the `typedMessageHash` before any signature validation and thus to pass in the same hashed payload to both the **EIP-1271** and `ecrecover` validation flows.

Offer Manual Review Findings

ORE-01M: Double Accounting of Native Funds

Type	Severity	Location
Logical Fault	Minor	Offer.sol:L254, L333

Description:

The `OfferFacet::unwrapNFT` function flow will invoke the `FundsLib::validateIncomingPayment` function twice which can result in double-accounting of incoming payments.

Impact:

The present mechanism cannot be exploited as the system does not adequately support native funds in its self-sale execution flow and the vulnerability would manifest solely when the `sellerDeposit` matched the `minimalPrice` in case of a self-sale.

Example:

contracts/protocol/facets/Offer.sol

SOL

```sol
229 // Check the caller is the the seller's assistant
230 {
231     uint256 sellerId = offer.sellerId;
232     EntityLib.validateSellerAssistantOrFacilitator(sellerId,
offer.facilitatorId);
233
234     handleBosonSellerDeposit(sellerId, exchangeToken, offer.sellerDeposit);
235 }
236
237 FermionStorage.ProtocolLookups storage pl = FermionStorage.protocolLookups();
238 address wrapperAddress = pl.fermionFNFTAddress[offerId];
```

Example (Cont.):

```sol

IBosonProtocol.PriceDiscovery memory _priceDiscovery;
_priceDiscovery.side = IBosonProtocol.Side.Wrapper;
_priceDiscovery.priceDiscoveryContract = wrapperAddress;
_priceDiscovery.conduit = wrapperAddress;
{
    uint256 bosonProtocolFee;
    if (_selfSale) {
        uint256 minimalPrice;
        (minimalPrice, bosonProtocolFee) = getMinimalPriceAndBosonProtocolFee(
            exchangeToken,
            offer.verifierFee,
            0
        );
        if (minimalPrice > 0) {
            FundsLib.validateIncomingPayment(exchangeToken, minimalPrice);
            IERC20(exchangeToken).safeTransfer(wrapperAddress, minimalPrice);
        }
```

Recommendation:

We advise the system to disallow payment of the seller deposit if it is in native form so as to prevent the system from treating a single `msg.value` deposit twice.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

An `if-revert` check was introduced to the **`OfferFacet::handleBosonSellerDeposit`** function ensuring that the Boson seller deposit has been pre-deposited if it is meant to be supplied via native funds, preventing the vulnerability from ever manifesting.

ORE-02M: Inexistent Support of Native Token

Type	Severity	Location
Logical Fault	Minor	Offer.sol:L255

Description:

The self-sale unwrapping flow does not adequately support native payments as the `minimalPrice` is being paid solely using an **EIP-20** transfer flow.

Impact:

The NFT unwrapping mechanisms are presently incompatible with native funds when a self-sale is unwrapped.

Example:

contracts/protocol/facets/Offer.sol

SOL

```sol
253 if (minimalPrice > 0) {
254     FundsLib.validateIncomingPayment(exchangeToken, minimalPrice);
255     IERC20(exchangeToken).safeTransfer(wrapperAddress, minimalPrice);
256 }
```

Recommendation:

We advise a native payment transfer flow to be properly supported.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The relevant segment was updated to transfer funds via the `FundsLib::transferFundsFromProtocol` function instead of a direct **EIP-20** transfer, ensuring native payments are properly supported in this particular execution flow.

ORE-03M: Seller-Defined Verification Time-Out

Type	Severity	Location
Logical Fault	Minor	Offer.sol:L291-L294

Description:

The overall execution flow of a RWA sale via the Fermion Protocol system involves a verification stage during which the asset is meant to be validated by a verifier to match the buyer's expectations.

The current system permits the seller rather than the buyer to set a verification time-out which can negatively impact the buyer in case of an inactive verifier.

Impact:

It is presently possible for the seller to negatively impact the buyer via a configuration they have sole control of.

Example:

contracts/protocol/facets/Offer.sol

SOL

```sol
291 uint256 itemVerificationTimeout = _verificationTimeout == 0
292     ? block.timestamp + FermionStorage.protocolConfig().verificationTimeout
293     : _verificationTimeout;
294 pl.itemVerificationTimeout[_tokenId] = itemVerificationTimeout;
```

Recommendation:

We advise the system to either set a verification time-out in a consensual way (i.e. where both the seller and buyer agree), or the default `verificationTimeout` configured to be applied in all circumstances.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

A system-wide maximum upper bound on the verification timeout has been imposed ensuring that the seller has limited control in terms of how high the verification timeout can be configured, effectively mitigating the effect up to which the seller can negatively impact their buyer.

As such, we consider this exhibit alleviated based on proper configuration and maintenance of the `maxVerificationTimeout` protocol variable.

ORE-04M: Incorrect Calculation of Minimal Price w/ Boson Protocol Fee

Type	Severity	Location
Mathematical Operations	Major	Offer.sol:L368

Description:

The referenced minimal price calculation is incorrect as it will attempt to grow the `_verifierFee` by the amount necessary so that the minimal price incorporates both the Boson Protocol fee and the verifier fee.

The calculation presently rounds downwards which would result in a `minimalPrice` permitted that is less than the `_verifierFee`.

This vulnerability results in a significant compromise of the system as the **`VerificationFacet::submitVerdictInternal`** function assumes the `offerPrice` is always greater than the `verifierFee` and performs an `unchecked` subtraction that would increase the available funds of either the buyer or the seller to be effectively unlimited.

Impact:

It is presently possible to supply a `minimalPrice` that would not satisfy the `_verifierFee`.

This results in an unchecked subtraction within **`VerificationFacet::submitVerdictInternal`** to result in an underflow that causes an unlimited fund availability increase to be processed under certain fee configurations.

Example:

contracts/protocol/facets/Offer.sol

SOL

```
368 minimalPrice = (HUNDRED_PERCENT * _verifierFee) / (HUNDRED_PERCENT -
bosonProtocolFeePercentage);
```

Recommendation:

We advise the calculation to round upwards, ensuring that the `_verifierFee` is always met.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

We extensively discussed this particular exhibit with the Fermion Protocol team which prompted them to perform a mathematical analysis of the conditions that the rounding error would manifest itself in.

The analysis which we independently validated showcased that the vulnerability would never manifest in the protocol due to the `itemPrice` being calculated as a subtraction rather than a percentage which acts as a "rounding-up" operation.

Given that the vulnerability is not applicable to the Fermion Protocol team's codebase, we consider it nullified and an item to keep track of in future implementations.

ReentrancyGuard Manual Review Findings

RGD-01M: Incorrect Re-Entrancy Guard Pattern

Type	Severity	Location
Logical Fault	Major	ReentrancyGuard.sol:L13-L29

Description:

The present `ReentrancyGuard::nonReentrant` function implementation will cause the re-entrancy guard to be validated solely when the contract is not invoking itself.

This represents a significant vulnerability as any nested call invoking the contract itself will result in the `GUARD_SLOT` being set to `0` before the upper-level transaction concludes, thereby invalidating the re-entrancy protection measure mid-execution which can pave the way for re-entrancy attacks even on `ReentrancyGuard::nonReentrant` functions.

Impact:

The current `ReentrancyGuard` flag can be reset to `0` during a re-entrant nested call initiated by the contract itself.

Example:

contracts/diamond/libraries/ReentrancyGuard.sol

SOL

```sol
13  modifier nonReentrant() {
14      if (msg.sender != address(this)) {
15          assembly {
16              if tload(GUARD_SLOT) {
17                  mstore(0, 0x6cee810b) // ReentrancyGuard.Reentrered.selector
18                  revert(0x1c, 0x04)
19              }
20              tstore(GUARD_SLOT, 1)
21          }
22      }
```

Example (Cont.):

```sol
        _;
        // Unlocks the guard, making the pattern composable.
        // After the function exits, it can be called again, even in the same transaction.
        assembly {
            tstore(GUARD_SLOT, 0)
        }
    }
```

Recommendation:

We advise the code to re-set the `GUARD_SLOT` to `0` solely when the `msg.sender` is not the contract itself (i.e. `address(this)`), preventing the outlined misbehaviour from manifesting.

To note, off-chain based invocations of `view` and `pure` calls do not apply the strict `staticcall` restrictions and thus this vulnerability can only be validated via a contract-to-contract interaction.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The code was updated per our recommendation, ensuring that the re-entrancy `GUARD_SLOT` is reset for non-intra-facet transactions and thus alleviating this exhibit in full.

SeaportWrapper Manual Review Findings

SWR-01M: Inexistent Restrictions of OpenSea Order (Fee)

Type	Severity	Location
Logical Fault	Minor	SeaportWrapper.sol:L77

Description:

The `SeaportWrapper::finalizeOpenSeaAuction` function is meant to accept an OpenSea Seaport order supplied to it via a top-level call, however, the function does not apply any restrictions on how the buyer order is defined.

In detail, the order is not validated to have a single `offer` and two considerations, and additionally does not restrict the `_openSeaFee`.

As a result, a user can set a very high OpenSea fee thereby griefing the seller of funds and can additionally set more considerations that they can fulfil via re-entrant calls during the transfer of offered assets as they are not restricted to the **EIP-20** item type.

Impact:

A malicious buyer can trick a seller into accepting an OpenSea offer that has a significantly high fee attached to it thereby causing them to receive significantly less funds than they expected to.

While off-chain mitigations for this type of attack should be in place, it is strongly advised to impose similar limitations on-chain to prevent automated systems from accepting improperly crafted orders.

Example:

contracts/protocol/clients/SeaportWrapper.sol

SOL

```sol
function finalizeOpenSeaAuction(
    uint256 _tokenId,
    SeaportTypes.AdvancedOrder calldata _buyerOrder
) internal returns (uint256 reducedPrice, address exchangeToken) {
    address wrappedVoucherOwner = _ownerOf(_tokenId); // tokenId can be taken from buyer order

    uint256 _price = _buyerOrder.parameters.offer[0].startAmount;
    uint256 _openSeaFee = _buyerOrder.parameters.consideration[1].startAmount;
    reducedPrice = _price - _openSeaFee;
```

Recommendation:

We advise a rudimentary level of OpenSea order sanitization to be introduced, particularly by restricting the `_openSeaFee` as a percentage of the overall `_price` and by ensuring two considerations and one offer has been set.

Alleviation (876c9921ae):

Input sanitization was introduced at the `OfferFacet` level and specifically the **`OfferFacet::unwrapNFT`** function that ensures the OpenSea fee has been configured to at most 2.5%, accounting for upward rounding fee values via a fixed addition of `1`.

As illustrated in the OpenSea documentation, the 2.5% fee is based **on the overall listing price** and not on the price that the seller receives which means that the current limitation mechanism is incorrect.

We advise the `OS_FEE_PERCENTAGE` limitation to be imposed on the **total price of the listing** (i.e. `_buyerOrder.parameters.offer[0].startAmount`), ensuring that the fee is properly restricted.

Alleviation (43f5ee8c55):

The code within **`OfferFacet::unwrapNFT`** was further updated to enforce the `OS_FEE_PERCENTAGE` check based on the total price of the listing as advised, alleviating this exhibit in full.

Verification Manual Review Findings

VNO-01M: Inexistent Restriction of Verification Timeout

Type	Severity	Location
Logical Fault	Minor	Verification.sol:L73-L84

Description:

The `VerificationFacet::changeVerificationTimeout` function will permit the verification time-out of a particular RWA to be extended, however, it is invoke-able by the seller and will ultimately hurt the buyer who is expecting the overall trade to succeed.

Impact:

The seller of a RWA can infinitely extend its verification which, in the case of an inactive verifier, will negatively impact the buyer.

Example:

contracts/protocol/facets/Verification.sol

SOL

```
61  /**
62   * @notice Change the verification timeout for a specific token
63   *
64   * Emits an ItemVerificationTimeoutChanged event
65   *
66   * Reverts if:
67   * - Verification region is paused
68   * - Caller is not the seller's assistant or facilitator
69   *
70   * @param _tokenId - the token ID
```

Example (Cont.):

```sol
    * @param _newTimeout - the new verification timeout
    */
    function changeVerificationTimeout(
        uint256 _tokenId,
        uint256 _newTimeout
    ) external notPaused(FermionTypes.PausableRegion.Verification) {
        (, FermionTypes.Offer storage offer) = FermionStorage.getOfferFromTokenId(_tokenId);

        EntityLib.validateSellerAssistantOrFacilitator(offer.sellerId, offer.facilitatorId);

        FermionStorage.protocolLookups().itemVerificationTimeout[_tokenId] = _newTimeout;

        emit ItemVerificationTimeoutChanged(_tokenId, _newTimeout);
    }
```

Recommendation:

We advise either the mechanism to be consensual (i.e. require approval by both the buyer and seller), or a maximum extension time-out to be imposed.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

A maximum verification timeout has been imposed per token ID, preventing the verification timeout from being infinitely extended and thus addressing the vulnerability described.

Access Code Style Findings

ASS-01C: Redundant Local Variable

Type	Severity	Location
Gas Optimization	Informational	Access.sol:L38, L40

Description:

The referenced `hasRole` local variable declaration is inefficient as the variable is utilized immediately after its declaration.

Example:

contracts/protocol/libs/Access.sol

SOL

```sol
38  bool hasRole = $._roles[_role].hasRole[account];
39
40  if (!hasRole) revert IAccessControl.AccessControlUnauthorizedAccount(account,
_role);
```

Recommendation:

We advise the `$._roles[_role].hasRole[account]` evaluation to be set in the `if` conditional directly thus optimizing the code's memory expansion needs.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The redundant local variable has been omitted as advised, optimizing the code's gas cost.

Common Code Style Findings

CNO-01C: Condition Short-Circuiting

Type	Severity	Location
Gas Optimization	Informational	Common.sol:L53

Description:

The `Common::checkStateAndCaller` function will inefficiently fetch the token's state before evaluating the `msg.sender` which is a significantly less expensive evaluation.

Example:

contracts/protocol/clients/Common.sol

SOL

```sol
52  FermionTypes.TokenState state = _getFermionCommonStorage().tokenState[_tokenId];
53  if (state != _expectedState || msg.sender != _expectedCaller) {
54      revert InvalidStateOrCaller(_tokenId, msg.sender, state);
55  }
```

Recommendation:

We advise the `if` condition's statement to evaluate the `msg.sender` prior to fetching end evaluating the token's state, optimizing the code's failure gas cost.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The Fermion Protocol team evaluated this exhibit and stated that the error yielded includes the `state` regardless, rendering the proposed optimization incorrect.

We concur with this analysis, and thus consider this exhibit nullified.

CNO-02C: Redundant Local Variable Declaration

Type	Severity	Location
Gas Optimization	Informational	Common.sol:L52, L53

Description:

The referenced local `state` variable declaration is inefficient as its value is immediately utilized in the ensuing `if` conditional.

Example:

contracts/protocol/clients/Common.sol

SOL

```
52  FermionTypes.TokenState state = _getFermionCommonStorage().tokenState[_tokenId];
53  if (state != _expectedState || msg.sender != _expectedCaller) {
```

Recommendation:

We advise the expression to replace the local variable's reference directly rendering the local declaration unnecessary and thus optimizing the code's memory expansion needs.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

Similarly to `CNO-01C`, this submission has been invalidated due to the `state` variable being used in the yielded `error`.

Config Code Style Findings

CGI-01C: Generic Typographic Mistake

Type	Severity	Location
Code Style	Informational	Config.sol:L108

Description:

The referenced line contains a typographical mistake (i.e. `private` variable without an underscore prefix) or generic documentational error (i.e. copy-paste) that should be corrected.

Example:

contracts/protocol/facets/Config.sol

```sol
108 // Make sure percentage is less than 10000
```

Recommendation:

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

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The referenced mistaken documentation line has been corrected, addressing this exhibit.

CGI-02C: Inefficient Initialization of Facet

Type	Severity	Location
Gas Optimization	Informational	Config.sol:L23-L28

Description:

The `ConfigFacet::init` function is inefficient as it will invoke several functions that will impose access control, pausability checks, and will invoke the `FermionStorage::protocolConfig` function multiple times redundantly.

Example:

contracts/protocol/facets/Config.sol

SOL

```sol
function init(FermionStorage.ProtocolConfig calldata _config) public {
    // Initialize protocol config params
    setTreasuryAddress(_config.treasury);
    setProtocolFeePercentage(_config.protocolFeePercentage);
    setVerificationTimeout(_config.verificationTimeout);
}
```

Recommendation:

We advise each configuration statement to be performed locally within the `ConfigFacet::init` function in an optimal way, significantly reducing the initialization gas cost of the `ConfigFacet`.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

All referenced functions have been refactored instead exposing an internal variant that does not impose unnecessary access control, thereby optimizing the codebase in an identical way.

Context Code Style Findings

CTX-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	Context.sol:L47

Description:

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

Example:

contracts/protocol/libs/Context.sol

SOL

```sol
46  if (msg.sender == protocolAddress && msgDataLength >= ADDRESS_LENGTH) {
47      return address(bytes20(msg.data[msgDataLength - ADDRESS_LENGTH:]));
48  } else {
```

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 (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The referenced operation has been wrapped in an `unchecked` code block as advised.

CustodyLib Code Style Findings

CLB-01C: Generic Typographic Mistake

Type	Severity	Location
Code Style	Informational	CustodyLib.sol:L107

Description:

The referenced line contains a typographical mistake (i.e. `private` variable without an underscore prefix) or generic documentational error (i.e. copy-paste) that should be corrected.

Example:

contracts/protocol/libs/CustodyLib.sol

SOL

```
107 revert CustodianVaultErrors.InssuficientBalanceToFractionalise(tokenId, diff);
```

Recommendation:

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

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The referenced typographic mistake has been corrected, addressing this exhibit.

CLB-02C: Tautological Assignment

Type	Severity	Location
Gas Optimization	Informational	CustodyLib.sol:L125

Description:

The referenced assignment is ineffectual.

Example:

contracts/protocol/libs/CustodyLib.sol

SOL

```sol
124 if (_externalCall && returnedAmount > 0) {
125     returnedAmount = returnedAmount;
126     FundsLib.transferFundsFromProtocol(exchangeToken, payable(msg.sender),
returnedAmount); // not using msgSender() since caller is FermionFNFT contract
127 }
```

Recommendation:

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

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The tautological assignment has been omitted from the codebase addressing this exhibit.

DiamondLoupeFacet Code Style Findings

DLF-01C: Redundant Variable Assignments

Type	Severity	Location
Gas Optimization	Informational	DiamondLoupeFacet.sol:L51, L122

Description:

The referenced assignments of `false` are redundant given that the `continue` statement that follows them will cause a re-declaration of the `continueLoop` variable and thus a reset of its value back to `false`.

Example:

contracts/diamond/facets/DiamondLoupeFacet.sol

SOL

```
39  bool continueLoop = false;
40  // find the functionSelectors array for selector and add selector to it
41  for (uint256 facetIndex; facetIndex < numFacets; facetIndex++) {
42      if (facets_[facetIndex].facetAddress == facetAddress_) {
43          facets_[facetIndex].functionSelectors[numFacetSelectors[facetIndex]] =
selector;
44          numFacetSelectors[facetIndex]++;
45          continueLoop = true;
46          break;
47      }
48  }
```

Example (Cont.):

```sol
// if functionSelectors array exists for selector then continue loop
if (continueLoop) {
    continueLoop = false;
    continue;
}
```

Recommendation:

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

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The referenced redundant assignments have been omitted, optimizing the code's gas cost.

Entity Code Style Findings

EYT-01C: Generic Typographic Mistake

Type	Severity	Location
Code Style	Informational	Entity.sol:L620

Description:

The referenced line contains a typographical mistake (i.e. `private` variable without an underscore prefix) or generic documentational error (i.e. copy-paste) that should be corrected.

Example:

contracts/protocol/facets/Entity.sol

SOL

```
620 function addOrRemoveFacilitatos(uint256 _sellerId, uint256[] calldata
_facilitatorIds, bool _add) internal {
```

Recommendation:

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

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The function's name was properly corrected, addressing this exhibit.

EntityLib Code Style Findings

ELB-01C: Non-Standard Invocation Style

Type	Severity	Location
Code Style	Informational	EntityLib.sol:L38

Description:

The `EntityLib::storeEntity` function invocation within the `EntityLib::createEntity` function is performed using explicit `EntityLib` notation which is non-standard.

Example:

contracts/protocol/libs/EntityLib.sol

SOL

```sol
38  EntityLib.storeEntity(entityId, _admin, newEntity, _roles, _metadata);
39  storeCompactWalletRole(entityId, _admin, 0xff << (31 * BYTE_SIZE), true, pl, pe);
// compact role for all current and potential future roles
40  emitAdminWalletAddedOrRemoved(entityId, _admin, true);
```

Recommendation:

We advise the function to be invoked directly similarly to the other function invocations.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The function is being invoked without the `EntityLib` specifier as advised, optimizing the code's legibility.

ELB-02C: Redundant Local Variable Declaration

Type	Severity	Location
Gas Optimization	Informational	EntityLib.sol:L294-L307, L309

Description:

The `EntityLib::validateSellerAssistantOrFacilitator` function will declare a local `isSellerOrFacilitator` variable that is consequently utilized in the immediate statement once.

Example:

contracts/protocol/libs/EntityLib.sol

SOL

```sol
294 bool isSellerOrFacilitator = hasWalletRole(
295     _sellerId,
296     _walletAddress,
297     FermionTypes.EntityRole.Seller,
298     FermionTypes.WalletRole.Assistant,
299     false
300 ) ||
301     hasWalletRole(
302         _facilitatorId,
303         _walletAddress,
```

Example (Cont.):

```sol
        FermionTypes.EntityRole.Seller,
        FermionTypes.WalletRole.Assistant,
        false
    );

if (!isSellerOrFacilitator) {
    revert EntityErrors.WalletHasNoRole(
        _sellerId,
        _walletAddress,
        FermionTypes.EntityRole.Seller,
        FermionTypes.WalletRole.Assistant
    );
}
```

Recommendation:

We advise the code to no longer declare a local variable and to utilize the assigned expression directly in the `if` block conditional, optimizing the code's memory expansion costs.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The redundant local variable declaration has been omitted as advised, optimizing the code's gas cost.

FermionFractions Code Style Findings

FFS-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	FermionFractions.sol:L284, L351, L721, L831

Description:

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

Example:

contracts/protocol/clients/FermionFractions.sol

SOL

```sol
279 if (_fractionAmount > votes.individual[msgSender]) {
280     revert NotEnoughLockedVotes(_tokenId, _fractionAmount,
votes.individual[msgSender]);
281 }
282 _transferFractions(address(this), msgSender, _fractionAmount);
283
284 votes.individual[msgSender] -= _fractionAmount;
```

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 (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

All referenced arithmetic operations have been safely wrapped in an `unchecked` code block as advised.

FFS-02C: Inefficient Conditional

Type	Severity	Location
Gas Optimization	Informational	FermionFractions.sol:L358

Description:

The referenced conditional will evaluate whether the `_fractions` of the user are non-zero yet transfer the `bidderFractions - lockedIndividualVotes` evaluation.

Example:

contracts/protocol/clients/FermionFractions.sol

SOL

```
358 if (_fractions > 0) _transferFractions(msgSender, address(this), bidderFractions - lockedIndividualVotes);
```

Recommendation:

We advise the `if` conditional to match the amount transferred, ensuring that the code is executed efficiently.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The overall logic was updated resulting in the `if` block's contents transferring the `_fractions` that are evaluated in its conditional thereby rendering this optimization inapplicable.

FermionWrapper Code Style Findings

FWR-01C: Redundant Re-Approval of Conduit

Type	Severity	Location
Gas Optimization	Informational	FermionWrapper.sol:L154

Description:

The `SeaportWrapper::wrapOpenSea` function needs to be invoked once as it will set an approval to the OpenSea conduit that cannot be consumed.

Example:

contracts/protocol/clients/FermionWrapper.sol

SOL

```sol
function wrap(uint256 _firstTokenId, uint256 _length, address _to) internal {
    for (uint256 i = 0; i < _length; i++) {
        uint256 tokenId = _firstTokenId + i;

        // Transfer vouchers to this contract
        // Not using safeTransferFrom since this contract is the recipient and we are sure it can handle the vouchers
        IERC721(voucherAddress).transferFrom(_msgSender(), address(this), tokenId);

        // Mint to the specified address
        _safeMint(_to, tokenId);
```

Example (Cont.):

```sol
            Common.changeTokenState(tokenId, FermionTypes.TokenState.Wrapped);
        }
        wrapOpenSea();
    }
```

Recommendation:

We advise the function to be invoked once during the contract's `FermionWrapper::initializeWrapper` call, significantly optimizing all `FermionWrapper::wrap` operations.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The referenced function is now invoked in the `FermionWrapper::initializeWrapper` hook, optimizing the code's operational gas cost.

Funds Code Style Findings

FSD-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	Funds.sol:L171, L177

Description:

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

Example:

contracts/protocol/facets/Funds.sol

SOL

```sol
168 if (_offset >= tokenCount) {
169     return new address[](0);
170 } else if (_offset + _limit > tokenCount) {
171     _limit = tokenCount - _offset;
172 }
```

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 (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The referenced operations have been wrapped in an `unchecked` code block each as advised.

Initialization Code Style Findings

INO-01C: Redundant Parenthesis Statement

Type	Severity	Location
Code Style	Informational	Initialization.sol:L204

Description:

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

Example:

contracts/protocol/facets/Initialization.sol

SOL

```
204 ds.supportedInterfaces[(_interfaces[i])] = _isSupported;
```

Recommendation:

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

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

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

INO-02C: Redundant Validation of Boson Protocol Address

Type	Severity	Location
Gas Optimization	Informational	Initialization.sol:L120

Description:

The referenced validation of the `_bosonProtocolAddress` is redundant as the ensuing `IBosonProtocol::getNextAccountId` function invocation would fail if it were zero.

Example:

contracts/protocol/facets/Initialization.sol

SOL

```sol
120 if (_bosonProtocolAddress == address(0) || _fermionFNFTImplementation == address(0))
121     revert FermionGeneralErrors.InvalidAddress();
122
123 IBosonProtocol bosonProtocol = IBosonProtocol(_bosonProtocolAddress);
124 uint256 bosonSellerId = bosonProtocol.getNextAccountId();
```

Recommendation:

We advise input validation to be omitted, optimizing the code's gas cost.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The redundant validation of the `_bosonProtocolAddress` variable has been omitted optimizing the code's gas cost.

MetaTransaction Code Style Findings

MTN-01C: Potentially Improper Error

Type	Severity	Location
Standard Conformity	Informational	MetaTransaction.sol:L337

Description:

The `MetaTransaction::verify` function will yield an `InvalidSignature` error in case the `magicValue` of the **EIP-1271** integration that is yielded does not equate the expected function selector.

Example:

contracts/protocol/facets/MetaTransaction.sol

SOL

```
334 try IERC1271(_user).isValidSignature(_hashedMetaTx, abi.encodePacked(_sigR,
_sigS, _sigV)) returns (
335     bytes4 magicValue
336 ) {
337     if (magicValue != IERC1271.isValidSignature.selector) revert InvalidSignature();
338     return true;
339 } catch {
```

Recommendation:

As such an error does not necessarily imply an invalid signature and might imply an expired one or similar, we advise the code to instead issue a `return` statement of `magicValue == IERC1271.isValidSignature.selector` thus permitting the **`MetaTransaction::executeMetaTransaction`** function to yield a `SignerAndSignatureDoNotMatch` error that might better depict the condition that resulted in the signature validation's failure.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The code was updated to instead yield a `SignatureValidationFailed` error which is a term applicable to both expired, malformed, and any similar signatures.

MTN-02C: Potentially Uncaught Error

Type	Severity	Location
Standard Conformity	Informational	MetaTransaction.sol:L334-L336

Description:

The referenced `try-catch` construct might result in a failure that is uncaught in case the `_user` is a smart contract that contains code, does not implement the `IERC1271::isValidSignature`, and contains a `fallback` function.

Specifically, a `try-catch` clause will **not** catch decoding errors in its `returns` declaration instead yielding an uncaught low-level error.

Impact:

As the code is not meant to continue execution in its `catch` clause, the impact of this exhibit is informational in nature as it relates to how off-chain software would handle such an error.

Example:

contracts/protocol/facets/MetaTransaction.sol

SOL

```
332 // Check if user is a contract implementing ERC1271
333 if (_user.code.length > 0) {
334     try IERC1271(_user).isValidSignature(_hashedMetaTx, abi.encodePacked(_sigR,
_sigS, _sigV)) returns (
335         bytes4 magicValue
336     ) {
337         if (magicValue != IERC1271.isValidSignature.selector) revert
InvalidSignature();
338         return true;
339     } catch {
340         revert InvalidSignature();
341     }
```

Example (Cont.):

```sol
342 }
```

Recommendation:

We advise the code to issue the `IERC1271::isValidSignature` function call using low-level constructs, ensuring that an inexistent return payload is adequately handled instead of resulting in a fatal uncaught error.

Alleviation (876c9921ae):

While the code was seemingly updated to accommodate for this issue, it has been improperly done so. In detail, it attempts to handle a return data payload in case of a failure in the `catch` clause which will still not be able to capture a decoding error.

We advise the relevant section of the **Solidity documentation to be consulted** and specifically the first note to be able to properly address this exhibit.

Alleviation (43f5ee8c55):

The code was updated to properly perform a manual low-level `staticcall` operation to the `_user`, permitting the contract to gracefully handle decoding errors in case the return data does not match the expected length.

As such, we consider this exhibit fully addressed.

MTN-03C: Redundant Function Implementation

Type	Severity	Location
Gas Optimization	Informational	MetaTransaction.sol:L192-L196

Description:

The `MetaTransaction::convertBytesToBytes4` function implementation is redundant given that a direct cast from the `bytes` data type to the `bytes4` data type is possible.

Example:

contracts/protocol/facets/MetaTransaction.sol

SOL

```
186 /**
187  * @notice Converts the given bytes to bytes4.
188  *
189  * @param _inBytes - the incoming bytes
190  * @return _outBytes4 -  The outgoing bytes4
191  */
192 function convertBytesToBytes4(bytes memory _inBytes) internal pure returns
(bytes4 _outBytes4) {
193     assembly {
194         _outBytes4 := mload(add(_inBytes, 32))
195     }
```

Example (Cont.):

```sol
196 }
```

Recommendation:

We advise such a cast to be performed rendering the `MetaTransaction::convertBytesToBytes4` unnecessary.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The redundant function implementation has been removed per our recommendation.

Pause Code Style Findings

PES-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	Pause.sol:L77, L80, L117

Description:

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

Example:

contracts/protocol/facets/Pause.sol

SOL

```
80  count++;
```

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 (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The referenced operations have been wrapped in an `unchecked` code block each as advised.

PES-02C: Redundant Local Variable

Type	Severity	Location
Gas Optimization	Informational	Pause.sol:L110, L117

Description:

The referenced local `region` variable is a constant memory expansion cost that is redundant given that the expression it is assigned to within the ensuing `for` loop is immediately used in the next statement.

Example:

contracts/protocol/facets/Pause.sol

SOL

```sol
110 uint256 region;
111 uint256 incomingPaused;
112
113 // Calculate the incoming paused status as the sum of individual regions
114 // Use "or" to get the correct value even if the same region is specified more
than once
115 for (uint256 i = 0; i < _regions.length; i++) {
116     // Get enum value as power of 2
117     region = 1 << uint256(_regions[i]);
118     incomingPaused |= region;
119 }
```

Recommendation:

We advise the local variable declaration to be omitted and the expression to be utilized directly in the `OR` assignment operator.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The redundant local variable has been omitted, optimizing the code's gas cost.

Storage Code Style Findings

SEG-01C: Inconsistent Storage Data Location Specifiers

Type	Severity	Location
Standard Conformity	Informational	Storage.sol:L12-L16

Description:

The referenced data location specifiers do not conform to any particular EIP despite the employment of **EIP-7201** name-spaced layouts in other contracts and dependencies.

Example:

contracts/protocol/libs/Storage.sol

SOL

```
12  bytes32 internal constant PROTOCOL_STATUS_POSITION =
keccak256("fermion.protocol.status");
13  bytes32 internal constant PROTOCOL_CONFIG_POSITION =
keccak256("fermion.protocol.config");
14  bytes32 internal constant PROTOCOL_ENTITIES_POSITION =
keccak256("fermion.protocol.entities");
15  bytes32 internal constant PROTOCOL_LOOKUPS_POSITION =
keccak256("fermion.protocol.lookups");
16  bytes32 internal constant META_TRANSACTION_POSITION =
keccak256("fermion.meta.transaction");
```

Recommendation:

We advise a standardized data location EIP to be utilized ensuring consistency across the codebase.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The referenced data locations were all updated to conform to the **EIP-7201** name-spaced layout, addressing this exhibit in full.

SEG-02C: Inefficient Distinct Data Structures

Type	Severity	Location
Gas Optimization	Informational	Storage.sol:L63, L67, L69, L71, L75, L77, L79, L81, L83, L85, L87, L89

Description:

The referenced `mapping` structures will utilize the same key to access different values which is inefficient.

Example:

contracts/protocol/libs/Storage.sol

SOL

```sol
62  // entity id => entity admin => pending status
63  mapping(uint256 => mapping(address => bool)) pendingEntityAdmin;
64  // offerId => wrapper address
65  mapping(uint256 => address) fermionFNFTAddress;
66  // tokenId => item price
67  mapping(uint256 => uint256) itemPrice;
68  // entity id => token address => amount
69  mapping(uint256 => mapping(address => uint256)) availableFunds;
70  // entity id => all tokens with balance > 0
71  mapping(uint256 => address[]) tokenList;
```

Recommendation:

We advise `mapping` declarations that utilize the same key to be combined into a single `mapping` pointing to a `struct`, greatly optimizing the code's gas cost when simultaneously accessing these data points.

Alleviation (876c9921ae66a350c14d92a9be5a3a87e9e67e1e):

The data structures have been combined to the greatest extent possible, greatly optimizing the gas costs involved in accessing them throughout the Fermion Protocol system.

Finding Types

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

Input Sanitization

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

Indeterminate Code

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

Language Specific

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

Curve Specific

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

Code Style

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

Mathematical Operations

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

Logical Fault

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

Privacy Concern

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

Proof Concern

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

Severity Definition

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

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

Severity Levels

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

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

Unknown Severity

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

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

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

Informational Severity

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

Minor Severity

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

Medium Severity

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

Major Severity

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

Likelihood & Impact Assessment

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

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

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

Disclaimer

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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