SEAPORT Security Review

Auditors

Sawmon and Natalie, Lead Security Researcher

Harikrishnan Mulackal, Lead Security Researcher

Dravee, Security Researcher

Ellahi, Junior Security Researcher

Alex Beregszaszi, Consultant

Report prepared by: Pablo Misirov

February 23, 2023

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Second version of Seaport, a marketplace protocol for safely and efficiently buying and selling NFTs.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of Seaport v1.2 according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 12 days in total, OpenSea engaged with Spearbit to review the Seaport protocol. In this period of time a total of **92** issues were found.

The original end date has been extended due holiday season.

Summary

Project Name	OpenSea
Repository	seaport
Commit	8d95af...
Type of Project	Marketplace protocol, NFT
Audit Timeline	Dec 13th - Jan 4th
Two week fix period	Jan 4th - Jan 18th

Issues Found

Severity	**Count**
Critical Risk	0
High Risk	0
Medium Risk	7
Low Risk	12
Gas Optimizations	19
Informational	54
Total	**92**

5 Findings

5.1 Medium Risk

5.1.1 The spent offer amounts provided to `OrderFulfilled` for collection of (advanced) orders is not the actual amount spent in general

Severity: Medium Risk

Context:

- OrderCombiner.sol#L455-L463
- OrderFulfiller.sol#L377-L385

Description: When `Seaport` is called to fulfill or match a collection of (advanced) orders, the `OrderFulfilled` is called before applying fulfillments and executing transfers. The offer and consideration items have the following forms:

$$C = (I_t, T, i, a_{curr}, R, a_{curr})$$

$$O = (I_t, T, i, a_{curr}, a_{curr})$$

Where

parameter	description
I_t	`itemType`
T	`token`
i	`identifier`
a_{curr}	the interpolation of `startAmount` and `endAmount` depending on the time and the fraction of the order.
R	consideration item's `recipient`
O	offer item.
C	consideration item.

The `SpentItem`s and `ReceivedItem` items provided to `OrderFulfilled` event ignore the last component of the offer/consideration items in the above form since they are redundant.

`Seaport` enforces that all consideration items are used. But for the endpoints in this context, we might end up with offer items with only a portion of their amounts being spent. So in the end $O.a_{curr}$ might not be the amount spent for this offer item, but `OrderFulfilled` emits $O.a_{curr}$ as the amount spent. This can cause discrepancies in off-chain bookkeeping by agents listening for this event.

The `fulfillOrder` and `fulfillAdvancedOrder` do not have this issue, since all items are enforced to be used. These two endpoints also differ from when there are collections of (advanced) orders, in that they would emit the `OrderFulfilled` at the of their call before clearing the reentrancy guard.

Recommendation: Make sure the accounting is updated to only provide the spent offer item amounts to `OrderFulfilled`. Moving the emission of this event to the end of the call flow, before clearing the reentrancy guard like the above mentioned simpler endpoint would make it easier to provide the correct values (and also would make the whole flow between different endpoints more consistent and potentially create an opportunity to refactor the codebase further).

Seaport: Fixed in PR 839 by making sure all unspent offer amounts are transferred to the recipient provided by the `msg.sender`.

Spearbit: Verified.

5.1.2 The spent offer item amounts shared with a `zone` for restricted (advanced) orders or with a contract `offerer` for orders of `CONTRACT` order type is not the actual spent amount in general

Severity: Medium Risk

Context:

- OrderCombiner.sol#L802-L807
- ConsiderationEncoder.sol#L440-L443
- ZoneInterface.sol#L13-L15
- ContractOffererInterface.sol#L16-L22
- OrderCombiner.sol#L322-L325
- FulfillmentApplier.sol#L299-L306
- FulfillmentApplier.sol#L432-L433
- FulfillmentApplier.sol#L120-L125
- OrderCombiner.sol#L794-L798

Description: When `Seaport` is called to fulfill or match a collection of (advanced) orders, there are scenarios where not all offer items will be used. When not all the current amount of an offer item is used and if this offer item belongs to an order which is of either `CONTRACT` order type or it is restricted order (and the `caller` is not the `zone`), then the spent amount shared with either the contract `offerer` or `zone` through their respective endpoints (`validateOrder` for `zone`s and `ratifyOrder` for contract `offerer`s) does not reflect the actual amount spent.

When `Seaport` is called through one of its more complex endpoints to match or fulfill orders, the offer items go through a few phases:

parameter	description
I_t	`itemType`
T	`token`
i	`identifier`
a_s	`startAmount`
a_e	`endAmount`
a_{curr}	the interpolation of `startAmount` and `endAmount` depending on the time and the fraction of the order.
O	offer item.

- Let's assume an offer item is originally $O = (I_t, T, i, a_s, a_e)$
- In `_validateOrdersAndPrepareToFulfill`, O gets transformed into $(I_t, T, i, a_{curr}, a_{curr})$
- Then depending on whether the order is part of a match (1, 2. 3) or fulfillment (1, 2) order and there is a corresponding fulfillment data pointing at this offer item, it might transform into (I_t, T, i, b, a_{curr}) where $b \in [0, \infty)$. For fulfilling a collection of orders $b \in \{0, a_{curr}\}$ depending on whether the offer item gets used or not, but for match orders, it can be in the more general range of $b \in [0, \infty)$.
- And finally for restricted or `CONTRACT` order types before calling `_assertRestrictedAdvancedOrderValidity`, the offer item would be transformed into $(I_t, T, i, a_{curr}, a_{curr})$.

So the `startAmount` of an `offer` item goes through the following flow:

$$a_s \to a_{curr} \to b \in [0, \infty) \to a_{curr}$$

And at the end a_{curr} is the amount used when `Seaport` calls into the `validateOrder` of a `zone` or `ratifyOrder` of a contract `offerer`. a_{curr} does not reflect the actual amount that this offer item has contributed to a combined amount used for an execution transfer.

Recommendation: For non-matched collection of (advanced) orders the actual spent amount for an offer item is $a_{curr} - b \in \{0, a_{curr}\}$ which basically reflects whether the item has been used or not.

So for these 2 `Seaport` endpoints (`fulfillAvailableOrders`, `fulfillAvailableAdvancedOrders`), one can calculate the actual spent amount.

For example, at the end of the flow for `startAmount`, one can do:

```
// Utilize assembly to calculate the spent amount.
assembly {
    let startAmountPtr := add(offerItem, Common_amount_offset)
    let originalAmount := mload(add(offerItem, Common_endAmount_offset))
    let unusedAmount := mload(startAmountPtr)

    mstore(
        startAmountPtr ,
        sub(originalAmount, unusedAmount)
    )
}
```

For matched orders since in certain scenarios, b can be any number in $\mathbb{B}_{256}$, it would be hard to say how much of that particular offer item was spent or not spent. And so the above suggestion would not work in general for matched orders.

Seaport: Fixed in PR 839 by making sure all unspent offer amounts are transferred to the recipient provided by the `msg.sender`.

Spearbit: Verified.

5.1.3 Empty `criteriaResolvers` for criteria-based contract orders

Severity: Medium Risk

Context:

- OrderValidator.sol#L312-L315
- CriteriaResolution.sol#L119

Description: There is a deviation in how criteria-based items are resolved for contract orders. For contract orders which have offers with criteria, the `_compareItems` function checks that the contract offerer returned a corresponding non-criteria based `itemType` when `identifierOrCriteria` for the original item is `0`, i.e., offering from an entire collection. Afterwards, the `orderParameters.offer` array is replaced by the `offer` array returned by the contract offerer.

For other criteria-based orders such as offers with `identifierOrCriteria = 0`, the `itemType` of the order is only updated during the criteria resolution step. This means that for such offers there should be a corresponding `CriteriaResolver` struct. See the following test:

```
modified   test/advanced.spec.ts
@@ -3568,9 +3568,8 @@ describe(`Advanced orders (Seaport v${VERSION})`, function () {
       // Seller approves marketplace contract to transfer NFTs
       await set1155ApprovalForAll(seller, marketplaceContract.address, true);

-      const { root, proofs } = merkleTree([nftId]);

-      const offer = [getTestItem1155WithCriteria(root, toBN(1), toBN(1))];
+      const offer = [getTestItem1155WithCriteria(toBN(0), toBN(1), toBN(1))];

       const consideration = [
         getItemETH(parseEther("10"), parseEther("10"), seller.address),
@@ -3578,8 +3577,9 @@ describe(`Advanced orders (Seaport v${VERSION})`, function () {
         getItemETH(parseEther("1"), parseEther("1"), owner.address),
       ];

+      // Replacing by `const criteriaResolvers = []` will revert
       const criteriaResolvers = [
-        buildResolver(0, 0, 0, nftId, proofs[nftId.toString()]),
+        buildResolver(0, 0, 0, nftId, []),
       ];

       const { order, orderHash, value } = await createOrder(
```

However, in case of contract offers with `identifierOrCriteria = 0`, Seaport 1.2 does not expect a corresponding `CriteriaResolver` struct and will revert if one is provided as the `itemType` was updated to be the corresponding non-criteria based `itemType`. See `advanced.spec.ts#L510` for a test case.

> *Note:* this also means that the fulfiller cannot explicitly provide the `identifier` when a contract order is being fulfilled.

A malicious contract may use this to their advantage. For example, assume that a contract offerer in Seaport only accepts criteria-based offers. The fulfiller may first call `previewOrder` where the criteria is always resolved to a rare NFT, but the actual execution would return an uninteresting NFT. If such offers also required a corresponding resolver (similar behaviour as regular criteria based orders), then this could be fixed by explicitly providing the `identifier`--akin to a slippage check.

In short, for regular criteria-based orders with `identifierOrCriteria = 0` the fulfiller can pick which `identifier` to receive by providing a `CriteriaResolver` (as long as it's valid). For contract orders, fulfillers don't have this option and contracts may be able to abuse this.

Recommendation: An alternative approach to criteria-based contract orders would be to remove the extra case in `_compareItems`. Now, contract offers will have to return the same `itemType` and `identifierOrCriteria` when a `generateOrder` call is made. However, this means that the fulfiller will be able to choose the `identifier` it wants to receive. This may not be the ideal in some cases, but it remains consistent with regular orders.

Seaport: We documented this deviation in PR 849.

Spearbit: Verified.

5.1.4 Advance orders of `CONTRACT` order types can generate orders with less consideration items that would break the aggregation routine

Severity: Medium Risk

Context:

- OrderValidator.sol#L444-L447
- FulfillmentApplier.sol#L561-L569

Description: When `Seaport` gets a collection of advanced orders to fulfill or match, if one of the orders has a `CONTRACT` order type, `Seaport` calls the `generateOrder(...)` endpoint of that order's `offerer`. `generateOrder(...)` can provide fewer consideration items for this order. So the total number of consideration items might be less than the ones provided by the `caller`.

But since the `caller` would need to provide the fulfillment data beforehand to `Seaport`, they might use indices that would turn to be out of range for the consideration in question after the modification applied for the contract `offerer` above. If this happens, the whole call will be reverted.

This issue is in the same category as *Advance orders of CONTRACT order types can generate orders with different consideration recipients that would break the aggregation routine*.

Recommendation: In order for the `caller` to be able to fulfill/match orders by figuring out how to aggregate and match different consideration and offer items, they would need to be able to have access to all the data before calling into `Seaport`. Contract `offerer`s are supposed to (it is not enforced currently) implement `previewOrder` which the `caller` can use before making a call to `Seaport`. But there is no guarantee that the data returned by `previewOrder` and `generateOrder` for the same shared inputs would be the same.

We can enforce that the contract `offerer` does not return fewer consideration items. If it needed to return less it can either revert or provide a `0` amount.

If the current conditions are going to stay the same, it is recommended to document this scenario and also provide more comments/documentation for `ContractOffererInterface`.

Seaport: Addressed in PR 842.

Spearbit: Verified.

5.1.5 `AdvancedOrder.numerator` and `AdvancedOrder.denominator` are unchecked for orders of `CONTRACT` order type

Severity: Medium Risk

Context:

- OrderValidator.sol#L150-L153
- OrderCombiner.sol#L455-L463

Description: For most advanced order types, we have the following check:

```
// Read numerator and denominator from memory and place on the stack.
uint256 numerator = uint256(advancedOrder.numerator);
uint256 denominator = uint256(advancedOrder.denominator);

// Ensure that the supplied numerator and denominator are valid.
if (numerator > denominator || numerator == 0) {
    _revertBadFraction();
}
```

For `CONTRACT` order types this check is skipped. For later calculations (calculating the current amount) `Seaport` uses the `numerator` and `denominator` returned by `_getGeneratedOrder` which as a pair it's either $(1, 1)$ or $(0, 0)$. `advancedOrder.numerator` is only used to skip certain operations in some loops when it is 0:

- Skip applying criteria resolvers.

- Skip aggregating the `amount` for executions.
- Skip the final validity check.

Skipping the above operations would make sense. But when for an `advancedOrder` with `CONTRACT` order type `_getGeneratedOrder` returns $(h, 1, 1)$ and `advancedOrder.numerator == 0`, we would skip applying criteria resolvers, aggregating the amounts from `offer` or `consideration` amounts for this order and skip the final validity check that would call into the `ratifyOrder` endpoint of the offerer. But emiting the following `OrderFulfilled` will not be skipped, even though this `advancedOrder` will not be used.

```
// Emit an OrderFulfilled event.
_emitOrderFulfilledEvent(
    orderHash,
    orderParameters.offerer,
    orderParameters.zone,
    recipient,
    orderParameters.offer,
    orderParameters.consideration
);
```

This can create discrepancies between what happens on chain and what off-chain agents index/record.

Recommendation: Even though `AdvancedOrder.numerator` and `AdvancedOrder.denominator` are not really used for advanced orders of `CONTRACT` type and `AdvancedOrder.numerator` is only used for signaling certain decision in the call, it would be best to either hoist the checks regarding these parameters to an earlier point:

```
// Read numerator and denominator from memory and place on the stack.
uint256 numerator = uint256(advancedOrder.numerator);
uint256 denominator = uint256(advancedOrder.denominator);

// Ensure that the supplied numerator and denominator are valid.
if (numerator > denominator || numerator == 0) {
    _revertBadFraction();
}

// If the order is a contract order, return the generated order.
if (orderParameters.orderType == OrderType.CONTRACT) {
    // Return the generated order based on the order params and the
    // provided extra data. If revertOnInvalid is true, the function
    // will revert if the input is invalid.
    return
        _getGeneratedOrder(
            orderParameters,
            advancedOrder.extraData,
            revertOnInvalid
        );
}
```

or for `orderParameters.orderType == OrderType.CONTRACT` enforce that `advancedOrder.numerator == advancedOrder.denominator == 1`.

Seaport: Fixed in PR 815.

Spearbit: Verified.

5.1.6 Calls to `PausableZone`'s `executeMatchAdvancedOrders` and `executeMatchOrders` would revert if unused native tokens would need to be returned

Severity: Medium Risk

Context:

- PausableZone.sol#L34
- PausableZone.sol#L149
- PausableZone.sol#L188
- OrderCombiner.sol#L704-L707

Description: In match (advanced) orders, one can provide native tokens as offer and consideration items. So a `PausableZone` would need to provide `msg.value` to call the corresponding `Seaport` endpoints. There are a few scenarios where not all the `msg.value` native tokens amount provided to the `Seaport` marketplace will be used:

1. Rounding errors in calculating the current amount of offer or consideration items. The `zone` can prevent sending extra native tokens to `Seaport` by pre-calculating these values and making sure to have its transaction to be included in the specific block that these values were calculated for (this is important when the start and end amount of an item are not equal).
2. The `zone` (un)intentionally sends more native tokens that it is necessary to `Seaport`.
3. The (advanced) orders sent for matching in `Seaport` include order type of `CONTRACT` offerer order and the offerer contract provides different amount for at least one item that would eventually make the whole transaction not use the full amount of `msg.value` provided to it.

In all these cases, since `PausableZone` does not have a `receive` or `fallback` endpoint to accept native tokens, when `Seaport` tries to send back the unsued native token amount the transaction may revert.

`PausableZone` not accepting native tokens:

```
$ export CODE=$(jq -r '.deployedBytecode' artifacts/contracts/zones/PausableZone.sol/PausableZone.json
↪  | tr -d '\n')
$ evm --code $CODE --value 1 --prestate genesis.json --sender
↪  0xb4d0000000000000000000000000000000000000 --nomemory=false --debug run

$ evm --input $(echo $CODE | head -c 44 - | sed -E s/0x//) disasm
6080806040526004908136101561001557600080fd
00000: PUSH1 0x80
00002: DUP1
00003: PUSH1 0x40
00005: MSTORE
00006: PUSH1 0x04
00008: SWAP1
00009: DUP2
0000a: CALLDATASIZE
0000b: LT
0000c: ISZERO
0000d: PUSH2 0x0015
00010: JUMPI
00011: PUSH1 0x00
00013: DUP1
00014: REVERT
```

trace of `evm ... --debug run`

```
 error: execution reverted
#### TRACE ####
PUSH1           pc=00000000 gas=4700000 cost=3

DUP1            pc=00000002 gas=4699997 cost=3
```

```
Stack:
00000000  0x80

PUSH1           pc=00000003 gas=4699994 cost=3
Stack:
00000000  0x80
00000001  0x80

MSTORE          pc=00000005 gas=4699991 cost=12
Stack:
00000000  0x40
00000001  0x80
00000002  0x80

PUSH1           pc=00000006 gas=4699979 cost=3
Stack:
00000000  0x80
Memory:
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 80  |................|

SWAP1           pc=00000008 gas=4699976 cost=3
Stack:
00000000  0x4
00000001  0x80
Memory:
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 80  |................|

DUP2            pc=00000009 gas=4699973 cost=3
Stack:
00000000  0x80
00000001  0x4
Memory:
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 80  |................|

CALLDATASIZE    pc=00000010 gas=4699970 cost=2
Stack:
00000000  0x4
00000001  0x80
00000002  0x4
Memory:
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 80  |................|
```

```
LT              pc=00000011 gas=4699968 cost=3
Stack:
00000000  0x0
00000001  0x4
00000002  0x80
00000003  0x4
Memory:
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 80  |................|

ISZERO          pc=00000012 gas=4699965 cost=3
Stack:
00000000  0x1
00000001  0x80
00000002  0x4
Memory:
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 80  |................|

PUSH2           pc=00000013 gas=4699962 cost=3
Stack:
00000000  0x0
00000001  0x80
00000002  0x4
Memory:
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 80  |................|

JUMPI           pc=00000016 gas=4699959 cost=10
Stack:
00000000  0x15
00000001  0x0
00000002  0x80
00000003  0x4
Memory:
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 80  |................|

PUSH1           pc=00000017 gas=4699949 cost=3
Stack:
00000000  0x80
00000001  0x4
Memory:
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

```
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 80  |................|

DUP1            pc=00000019 gas=4699946 cost=3
Stack:
00000000  0x0
00000001  0x80
00000002  0x4
Memory:
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 80  |................|

REVERT          pc=00000020 gas=4699943 cost=0
Stack:
00000000  0x0
00000001  0x0
00000002  0x80
00000003  0x4
Memory:
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 80  |................|

#### LOGS ####
```

genesis.json:

```
{
  "gasLimit": "4700000",
  "difficulty": "1",
  "alloc": {
    "0xb4d0000000000000000000000000000000000000": {
      "balance": "1000000000000000000000000",
      "code": "",
      "storage": {}
    }
  }
}
```

```
// file: test/zone.spec.ts
...
  it("Fulfills an order with executeMatchAdvancedOrders with NATIVE Consideration Item", async () => {
    const pausableZoneControllerFactory = await ethers.getContractFactory(
      "PausableZoneController",
      owner
    );
    const pausableZoneController = await pausableZoneControllerFactory.deploy(
      owner.address
    );

    // Deploy pausable zone
    const zoneAddr = await createZone(pausableZoneController);
```

```
// Mint NFTs for use in orders
const nftId = await mintAndApprove721(seller, marketplaceContract.address);

// Define orders
const offerOne = [
  getTestItem721(nftId, toBN(1), toBN(1), undefined, testERC721.address),
];
const considerationOne = [
  getOfferOrConsiderationItem(
    0,
    ethers.constants.AddressZero,
    toBN(0),
    parseEther("0.01"),
    parseEther("0.01"),
    seller.address
  ),
];
const { order: orderOne, orderHash: orderHashOne } = await createOrder(
  seller,
  zoneAddr,
  offerOne,
  considerationOne,
  2
);

const offerTwo = [
  getOfferOrConsiderationItem(
    0,
    ethers.constants.AddressZero,
    toBN(0),
    parseEther("0.01"),
    parseEther("0.01"),
    undefined
  ),
];
const considerationTwo = [
  getTestItem721(
    nftId,
    toBN(1),
    toBN(1),
    buyer.address,
    testERC721.address
  ),
];
const { order: orderTwo, orderHash: orderHashTwo } = await createOrder(
  buyer,
  zoneAddr,
  offerTwo,
  considerationTwo,
  2
);

const fulfillments = [
  [[[0, 0]], [[1, 0]]],
  [[[1, 0]], [[0, 0]]],
].map(([offerArr, considerationArr]) =>
  toFulfillment(offerArr, considerationArr)
);

// Perform the match advanced orders with zone
const tx = await pausableZoneController
  .connect(owner)
```

```
      .executeMatchAdvancedOrders(
        zoneAddr,
        marketplaceContract.address,
        [orderOne, orderTwo],
        [],
        fulfillments,
        { value: parseEther("0.01").add(1) } // the extra 1 wei reverts the tx
      );

    // Decode all events and get the order hashes
    const orderFulfilledEvents = await decodeEvents(tx, [
      { eventName: "OrderFulfilled", contract: marketplaceContract },
    ]);
    expect(orderFulfilledEvents.length).to.equal(fulfillments.length);

    // Check that the actual order hashes match those from the events, in order
    const actualOrderHashes = [orderHashOne, orderHashTwo];
    orderFulfilledEvents.forEach((orderFulfilledEvent, i) =>
      expect(orderFulfilledEvent.data.orderHash).to.be.equal(
        actualOrderHashes[i]
      )
    );
  });
...
```

This bug also applies to `Seaport 1.1` and `PausableZone` (0x004C00500000aD104D7DBd00e3ae0A5C00560C00)

Recommendation: It is really important for `zone`s that are trying to match orders that would involve native tokens to be able to receive those tokens back from `Seaport` if all of them are not used. In case of `Solidity` contracts, one should define `receive` or `fallback` endpoints for these contracts (or the `__default__` function if using `Vyper`).

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.1.7 ABI decoding for bytes: `memory` can be corrupted by maliciously constructing the `calldata`

Severity: Medium Risk

Context: ConsiderationDecoder.sol#L51-L62

Description: In the code snippet below, `size` can be made `0` by maliciously crafting the `calldata`. In this case, the free memory is not incremented.

```
assembly {
    mPtrLength := mload(0x40)
    let size := and(
        add(
            and(calldataload(cdPtrLength), OffsetOrLengthMask),
            AlmostTwoWords
        ),
        OnlyFullWordMask
    )
    calldatacopy(mPtrLength, cdPtrLength, size)
    mstore(0x40, add(mPtrLength, size))
}
```

This has two different consequences:

1. If the memory offset `mPtrLength` is immediately used then junk values at that memory location can be interpreted as the decoded `bytes` type. In the case of Seaport 1.2, the likelihood of the current free memory pointing to junk value is low. So, this case has low severity.

2. The consequent memory allocation will also use the value `mPtrLength` to store data in `memory`. This can lead to corrupting the initial memory data. In the worst case, the next allocation can be tuned so that the first bytes data can be any arbitrary data.

To make the `size` calculation return `0`:

1. Find a function call which has `bytes` as a (nested) parameter.
2. Modify the `calldata` field where the length of the above byte is stored to the new length `0xffffe0`.
3. The calculation will now return `size = 0`.

Note: there is an additional requirement that this `bytes` type should be inside a dynamic struct. Otherwise, for example, in case of `function foo(bytes calldata signature)` , the compiler will insert a check that `calldatasize` is big enough to fit `signature.length`. Since the value `0xffffe0` is too big to be fit into `calldata`, such an attack is impractical.

However, for `bytes` type inside a dynamic type, for example in `function foo(bytes[] calldata signature)`, this check is skipped by solc (likely because it's expensive). For a practical exploit we need to look for such function. In case of Seaport 1.2 this could be the `matchAdvancedOrders(AdvancedOrder[] calldata orders, ...)` function. The struct `AdvancedOrder` has a nested parameter `bytes signature` as well as `bytes extraData`.

In the above exploit one would be able to maliciously modify the `calldata` in such a way that Seaport would interpret the data in `extraData` as the signature. Here is a proof of concept for a simplified case that showcases injecting an arbitrary value into a decoded `bytes`.

As for severity, even though interpreting `calldata` differently may not fundamentally break the protocol, an attacker with enough effort may be able to use this for subtle phishing attacks or as a precursor to other attacks.

Recommendation: Updating `OnlyFullWordMask` to `0xff_ff_ff_e0` will not fix this as you can still replace `len` by `0xff_ff_ff_e0` and get the same effect. Also see *The size calculation can be incorrect for large numbers*.

Seaport: Fixed in PR 789.

Spearbit: Verified.

5.2 Low Risk

5.2.1 Advance orders of `CONTRACT` order types can generate orders with different consideration recipients that would break the aggregation routine

Severity: Low Risk

Context:

- ConsiderationDecoder.sol#L569-L574
- FulfillmentApplier.sol#L722-L736

Description: When `Seaport` receives a collection of advanced orders to match or fulfill, if one of the orders has a `CONTRACT` order type, `Seaport` calls the `generateOrder(...)` endpoint of that order's `offerer`. `generateOrder(...)` can provide new consideration item recipients for this order. These new recipients are going to be used for this order from this point on. In `_getGeneratedOrder`, there is no comparison between old or new consideration recipients.

The provided new recipients can create an issue when aggregating consideration items. Since the fulfillment data is provided beforehand by the caller of the `Seaport` endpoint, the caller might have provided fulfillment aggregation data that would have aggregated/combined one of the consideration items of this changed advance order with another consideration item. But the aggregation had taken into consideration the original recipient of the order in question. Multiple consideration items can only be aggregated if they share the same `itemType`, `token`, `identifier`, and `recipient` (ref). The new recipients provided by the contract offerer can break this invariant and in turn cause a revert.

Recommendation: Either

- The original consideration item recipients would need to be shared with the contract `offerer` when `generateOrder(...)` is called and they would stay the same for the new consideration items. This way the `offerer` can check these recipients and revert the call if needed. In this case, the caller of `Seaport` endpoints would need to somehow (perhaps using the `previewOrder` endpoint) get the recipients beforehand.
- Ensure that the old and new consideration recipients are the same.

Additionally and if changes are not applied, consider documenting this scenario for the users you call into `Seaport` or create custom offerer contracts. Adding more comments/documentation for `previewOrder` endpoint and `generateOrder` is also recommended.

Seaport: Fixed in PR 824 which ensures that either the new recipient can be any address if the original was `address(0)` or the new and old consideration recipients have to match (otherwise the call would revert).

Spearbit: Verified.

5.2.2 `CriteriaResolvers.criteriaProof` is not validated in the `identifierOrCriteria == 0` case

Severity: Low Risk

Context: CriteriaResolution.sol#L199-L206

Description: In the case of `identifierOrCriteria == 0`, the criteria resolver completely skips any validations on the Merkle proof and in particular is missing the validation that `CriteriaResolvers.criteriaProof.length == 0`.

Note: This is also present in Seaport 1.1 and may be a known issue.

Proof of concept:

```
modified   test/advanced.spec.ts
@@ -3568,9 +3568,8 @@ describe(`Advanced orders (Seaport v${VERSION})`, function () {
       // Seller approves marketplace contract to transfer NFTs
       await set1155ApprovalForAll(seller, marketplaceContract.address, true);

-      const { root, proofs } = merkleTree([nftId]);

-      const offer = [getTestItem1155WithCriteria(root, toBN(1), toBN(1))];
+      const offer = [getTestItem1155WithCriteria(toBN(0), toBN(1), toBN(1))];

       const consideration = [
         getItemETH(parseEther("10"), parseEther("10"), seller.address),
@@ -3578,8 +3577,9 @@ describe(`Advanced orders (Seaport v${VERSION})`, function () {
         getItemETH(parseEther("1"), parseEther("1"), owner.address),
       ];

+      // Add a junk criteria proof and the test still passes
       const criteriaResolvers = [
-        buildResolver(0, 0, 0, nftId, proofs[nftId.toString()]),
+        buildResolver(0, 0, 0, nftId,
↪  ["0xdead000000000000000000000000000000000000000000000000000000000000"]),
       ];

       const { order, orderHash, value } = await createOrder(
```

Recommendation: Consider adding the following additional check:

```
modified   contracts/lib/CriteriaResolution.sol
@@ -203,6 +203,8 @@ contract CriteriaResolution is CriteriaResolutionErrors {
                 identifierOrCriteria,
                 criteriaResolver.criteriaProof
             );
+        } else {
+            require(criteriaResolver.criteriaProof.length == 0);
         }

         // Update item type to remove criteria usage.
```

Seaport: Fixed in PR 825.

Spearbit: Verified.

5.2.3 Calls to `TypehashDirectory` will be successful

Severity: Low Risk

Context:

- TypehashDirectory.sol#L119

Description: `TypehashDirectory`'s deployed bytecode starts with `00` which corresponds to `STOP` opcode (`SSTORE2` also uses this pattern). This choice for the 1st bytecode causes accidental calls to the contract to succeed silently.

Recommendation: Document the reason why `STOP` was used for the 1st opcode. If it's not necessary to have `STOP` to be the first opcode, use an opcode that reverts calls to `TypehashDirectory` as it is only used as a data storage contract.

Seaport: Fixed in PR 799.

Spearbit: Verified.

5.2.4 `_isValidBulkOrderSize` does not perform the `signature` length validation correctly.

Severity: Low Risk

Context:

- Verifiers.sol#L122-L125

Description: In `_isValidBulkOrderSize` the `signature`'s length validation is performed as follows:

```
let length := mload(signature)
validLength := and(
    lt(length, BulkOrderProof_excessSize),
    lt(and(sub(length, BulkOrderProof_minSize), AlmostOneWord), 2)
)
```

The `sub` opcode in the above snippet wraps around. If this was the correct formula then it would actually simplify to:

```
lt(and(sub(length, 3), AlmostOneWord), 2)
```

The simplified and the current version would allow `length` to also be $3, 4, 35, 36, 67, 68$ but `_isValidBulkOrderSize` actually needs to check that `length` (l) has the following form:

$$l = (64 + x) + 3 + 32y$$

where $x \in \{0, 1\}$ and $y \in \{1, 2, \cdots, 24\}$ (y represents the height/depth of the bulk order).

Recommendation: Modify the `assembly` block to reflect the constraints needed for the above formula:

```
let z := sub(mload(signature), BulkOrderProof_minSize)
validLength := and(
   lt(z, 738), // 738 = (1 + 32 * 23) + 1, named constant BulkOrderProof_rangeSize
   lt(and(z, AlmostOneWord), 2)
)
```

- Verification:

`lt(sub(length, 99), 738)`

$$l - 99 = l - (64 + 3 + 32) = x + 32(y - 1) \leq 1 + 32 \cdot 23 = 737 < 738$$

This also takes care of underflows and we end up with a condition that $l \in \{99, 100, \cdots, 836\}$

`lt(and(add(length, 29), 31), 2)`

$$(l + \texttt{0b11101})\&\texttt{0b11111} \in \{0, 1\}$$

translates into $l + 29 \equiv 0, 1 (\text{mod} 32)$ or $l \equiv 3, 4 (\text{mod} 32)$ or $l - 99 \equiv 0, 1 (\text{mod} 32)$ which enforces $l - 99$ to be of a form $x + 32y'$ where $x \in \{0, 1\}$ and $y' \in \{Z\}$.

From the first part we know that $l - 99 \in \{0, 1, \cdots, 737\}$ and so that would restrict y' to be in $\{0, 1, \cdots, 23\}$

And so

$$l = (64 + x) + 3 + 32(y' + 1) \in 67 + \{0, 1\} + 32\{1, 2, \cdots, 24\}$$

Spearbit: The solution mentioned above might be cheaper than PR 797 (depends on the stack juggling by the compiler).

Seaport: Leaving it as-is for 1.2 as it's close either way.

Spearbit: Verified.

5.2.5 When `contractNonce` occupies more than `12` bytes the truncated `nonce` shared back with the contract `offerer` through `ratifyOrder` would be smaller than the actual stored nonce

Severity: Low Risk

Context:

- ConsiderationEncoder.sol#L146
- OrderValidator.sol#L385-L387

Description: When `contractNonce` occupies more than `12` bytes the truncated `nonce` shared back with the contract `offerer` through `ratifyOrder` would be smaller than the actual stored nonce:

```
// Write contractNonce to calldata
dstHead.offset(ratifyOrder_contractNonce_offset).write(
    uint96(uint256(orderHash))
);
```

This is due to the way the `contractNonce` and the `offerer`'s address are mixed in the `orderHash`:

```
assembly {
    orderHash := or(contractNonce, shl(0x60, offerer))
}
```

Recommendation: One can avoid the truncation by using `XOR` when calculating the `orderHash`:

```
orderHash := xor(contractNonce, offerer)
```

and sending the full `orderHash` to the contract offerer's `ratifyOrder` endpoint:

```
dstHead.offset(ratifyOrder_contractNonce_offset).write(orderHash);
```

The contract offerer can deduct the `nonce` by `XOR`ing its address with the received hash again:

```
nonce := xor(orderHash, address())
```

This would also make the calculations cheaper.

Seaport: Fixed in commit f82012.

Spearbit: Verified.

5.2.6 `abi_decode_bytes` does not mask the copied data length

Severity: Low Risk

Context:

- ConsiderationDecoder.sol#L60

Description: When `abi_decode_bytes` decodes `bytes`, it does not mask the copied length of the data in memory (other places where the length is masked by `OffsetOrLengthMask`).

Recommendation: Make sure to also mask the copied length before saving it to the memory.

Seaport: Fixed in PR 823.

Spearbit: Verified.

5.2.7 `OrderHash` in the context of contract orders need not refer to a unique order

Severity: Low Risk

Context: OrderValidator.sol#L386

Description: In Seaport 1.1 and in Seaport 1.2 for non-contract orders, order hashes have a unique correspondence with the order, i.e., it can be used to identify the status of an order on-chain and track it.

However, in case of contract orders, this is not the case. It is simply the current nonce of the `offerer`, combined with the address. This cannot be used to uniquely track an order on-chain.

```
uint256 contractNonce;
unchecked {
    contractNonce = _contractNonces[offerer]++;
}

assembly {
    orderHash := or(contractNonce, shl(0x60, offerer))
}
```

Here are some example scenarios where this can be problematic:

Scenario 1: A reverted contract order and the adjacent succeeding contract order will have the same order hash, regardless of whether they correspond to the same order.

1. Consider Alice calling `fulfilledAdvancedOrder` for a contract order with `offerer = X`, where `X` is a smart contract that offers contract orders on Seaport 1.2. Assume that this transaction failed because enough gas was not provided for the `generateOrder` call. This tx would revert with a custom error `InvalidContractOrder`, generated from `OrderValidator.sol#L391`.

2. Consider `Bob` calling `fulfilledAdvancedOrder` for a different contract order with `offerer = X`, the same smart contract offerer. This order will succeed and emit the `OrderFulfilled` event from `OrderFulfiller.sol#L124`

In the above scenario, there are two different orders, one that reverted on-chain and the other that succeeded, both having the same `orderHash` despite the orders only sharing the same contract `offerer`--the other parameters can be completely arbitrary.

Scenario 2: Contract order hashes computed off-chain can be misleading.

1. Consider Alice calling `fulfilledAdvancedOrder` for a contract order with `offerer = X`, where `X` is a smart contract that offers contract orders on Seaport 1.2. Alice computed the `orderHash` of their order off-chain by simulating the transaction, sends the transaction and polls the `OrderFulfilled` event with the same `orderHash` to know if the order has been fulfilled.
2. Consider Bob calling `fulfilledAdvancedOrder` for any contract order with `offerer = X`, the same smart contract offerer.
3. Bob's transaction gets included first. An `OrderFulfilled` event is emitted, with the `orderHash` to be the same hash that Alice computed off-chain! Alice may believe that their order succeeded.

Note: for non-contract Orders, the above approach would be valid, i.e., one may generate and sign an order, compute the order hash of an order off-chain and poll for an `OrderFulfilled` with the order hash to know that it was fulfilled. *Note*: even though there is an easier way to track if the order succeeded in these cases, in the general case, Alice or Bob need not be the one executing the orders on-chain. And an off-chain agent may send misleading notifications to either parties that their order succeeded due to this quirk with contract order hashes.

Recommendation:

1. Consider computing the order hash for contract orders similarly to how regular orders are hashed.
2. If the same mechanism is maintained, document that using `orderHashes` as a unique identifier is not reliable.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.2.8 When `_contractNonces[offerer]` gets updated no event is emitted

Severity: Low Risk

Context:

- OrderValidator.sol#L382
- CounterManager.sol#L54

Description: When `_contractNonces[offerer]` gets updated no event is emitted. This is in contrast to when a `counter` is updated.

One might be able to extract the `_contractNonces[offerer]` (if it doesn't overflow `12` bytes to enter into the `offerer` region in the `orderhash`) from a later event when `OrderFulfilled` gets emited. `OrderFulfilled` only gets emitted for an order of `CONTRACT` type if the `generateOrder(...)`'s return data satisffies all the constraints.

Recommendation: Emit a custom event when `_contractNonces[offerer]` gets updated if it's important for off-chain agents to monitor this value.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.2.9 In general a contract `offerer` or a `zone` cannot draw a conclusion accurately based on the spent offer amounts or received consideration amounts shared with them post-trasnfer

Severity: Low Risk

Context:

- ContractOffererInterface.sol#L7
- ContractOffererInterface.sol#L16
- ZoneInterface.sol#L13
- ZoneInteraction.sol#L92

Description: When one calls one of the `Seaport` endpoints that fulfills or matches a collection of (advanced) orders, the used offer or consideration items will go through different modification steps in the memory. In particular, the `startAmount` a of these items is an important parameter to inspect:

$$a \rightarrow a' \rightarrow b \rightarrow a'$$

a : original `startAmount` parameter shared to `Seaport` by the caller encoded in the memory. a' : the interpolated value and for orders of `CONTRACT` order type it is the value returned by the contract `offerer` (interpolation does not have an effect in this case since the `startAmount` and `endAmount` are enforced to be equal). b : must be 0 for used consideration items, otherwise the call would revert. For offer items, it can be in $[0, \infty)$ (See *The spent offer item amounts shared with a `zone` for restricted (advanced) orders or with a contract `offerer` for orders of `CONTRACT` order type is not the actual spent amount in general*).

a' : is the final amount shared by `Seaport` to either a `zone` for restricted orders and a contract `offerer` for `CONTRACT` order types.

- Offer Items

For offer items, perhaps the `zone` or the contract `offerer` would like to check that the `offerer` has spent a maximum a' of that specific offer item. For the case of restricted orders where the `zone`'s `validateOrder(...)` will be called, the `offerer` might end up spending more than a' amount of a specific token with the same identifier if the collection of orders includes:

- A mix of open and restricted orders.
- Multiple `zone`s for the same offerer, offering the same token with the same identifier.
- Multiple orders using the same `zone`. In this case, the `zone` might not have a sense of the orders of the transfers or which orders are included in the transaction in question (unless the contexts used by the `zone` enforces the exact ordering and number of items that can be matched/fulfilled in the same transaction). Note the order of transfers can be manipulated/engineered by constructing specific fulfillment data. Given a fulfillment data to combine/aggregate orders, there could be permutations of it that create different ordering of the executions.
- An order with an actor (a consideration recipient, contract offerer, weird token, ...) that has approval to transfer this specific offer item for the `offerer` in question. And when `Seaport` calls into (`NATIVE`, `ERC1155` token transfers, ...) this actor, the actor would transfer the token to a different address than the `offerer`.

There also is a special case where an order with the same offer item token and identifier is signed on a different instance of `Seaport` (1.0, 1.1, 1.2, ..., or other non-official versions) which an actor (a consideration recipient, contract offerer, weird token, ...) can cross-call into (related *Cross-Seaport re-entrancy with the stateful `validateOrder` call*).

The above issue can be avoided if the `offerer` makes sure to not sign different transactions across different or the same instances of `Seaport` which

1. Share the same offer type, offer token, and offer identifier,
2. but differ in a mix of zone, and order type

3. can be active at a shared `timestamp`

And/or the `offerer` does not give untrusted parties their token approvals.

A similar issue can arise for a contract `offerer` if they use a mix of signed orders of non-`CONTRACT` order type and `CONTRACT` order types.

- Consideration Items

For consideration items, perhaps the `zone` or the contract `offerer` would like to check that the `recipient` of each consideration item has received a minimum of *a'* of that specific consideration item. This case also is similar to the offer items issues above when a mix of orders has been used.

Recommendation: The above issues and notes should be documented for the users. Document the decision for the current `zone` and contract `offerer` interaction patterns.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.2.10 Cross-Seaport re-entrancy with the stateful `validateOrder` call

Severity: Low Risk

Context: BasicOrderFulfiller.sol#L280

Description: The re-entrancy check in Seaport 1.2 will prevent the Zone from interacting with Seaport 1.2 again. However, an interesting scenario would happen when if the conduit has open channels to both Seaport 1.1 and Seaport 1.2 (or different deployments/forks of Seaport 1.2).

This can lead to cross Seaport re-entrancy. This is not immediately problematic as Zones have limited functionality currently. But since Zones can be as flexible as possible, Zones need to be careful if they can interact with multiple versions of Seaport.

Note: for Seaport 1.1's zone, the check `_assertRestrictedBasicOrderValidity` happens before the transfers, and it's also a `staticcall`.

In the future, Seaport 1.3 could also have the same zone interaction, i.e., stateful calls to zones allowing for complex cross-Seaport re-entrancy between 1.2 and 1.3.

Note: also see *getOrderStatus and getContractOffererNonce are prone to view reentrancy* for concerns around view-only re-entrancy.

Recommendation:

- Document that cross-Seaport re-entrancy can be possible in general, and potentially problematic when paired with the `_assertRestricedBasicOrderValidity` / `validateOrder`, and when the conduit has open channels with multiple versions of Seaport.
- Avoiding having open channels to both Seaport 1.1, Seaport 1.2 at the same time, or between multiple versions of Seaport.
- For Zones with complex logic, consider deploying a new Conduit and carefully consider the risk of opening new channels.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.2.11 `getOrderStatus` and `getContractOffererNonce` are prone to view reentrancy

Severity: Low Risk

Context:

- Consideration.sol#L548
- Consideration.sol#L607
- OrderValidator.sol#L84-L87
- OrderValidator.sol#L277-L280
- OrderValidator.sol#L284-L287
- OrderValidator.sol#L382

Description: In a `Consideration` or the `Seaport` contract, once `_orderStatus[orderHash]` or `_contractNonces[offerer]` gets updated if there is a mix of contract offerer orders and partial orders are used, `Seaport` would call into the offerer contracts (let's call one of these offerer contracts *X*). In turn *X* can be a contract that would call into other contracts (let's call them *Y*) that take into consideration `_orderStatus[orderHash]` or `_contractNonces[offerer]` in their codebase by calling `getOrderStatus` or `getContractOffererNonce`

The values for `_orderStatus[orderHash]` or `_contractNonces[offerer]` might get updated after *Y* seeks those from `Seaport` due to for example multiple partial orders with the same `orderHash` or multiple offerer contract orders using the same offerer. Therefore *Y* would only take into consideration the mid-flight values and not the final ones after the whole transaction with `Seaport` is completed.

Recommendation: We need to either make sure to update the storage parameters at the end of the call to `Seaport` order fulfilling/matching endpoints (after all the calls to external contracts) or add `_assertNonReentrant()` guard to the `getOrderStatus` and `getContractOffererNonce` endpoints to avoid other contracts *Y* to read mid-flight storage parameters.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.2.12 The size calculation can be incorrect for large numbers

Context: `ConsiderationEncoder.sol#L53-L63`, `ConsiderationConstants.sol#L113`

Description: The maximum value of memory offset is defined in `PointerLibraries.sol#L22` as `OffsetOrLengthMask = 0xffffffff`, i.e., $2^{32} - 1$. However, the mask `OnlyFullWordMask = 0xffffe0;` is defined to be a 24-bit number.

Assume that the length of the `bytes` type where `src` points is `0xffffe0`, then the following piece of code incorrectly computes the size as `0`.

```
function abi_encode_bytes(
    MemoryPointer src,
    MemoryPointer dst
) internal view returns (uint256 size) {
    unchecked {
        size =
            ((src.readUint256() & OffsetOrLengthMask) + AlmostTwoWords) &
            OnlyFullWordMask;
        ...
```

This is because the constant `OnlyFullWordMask` does not have the two higher order bytes set (as a 32-bit type).

Note: in practice, it can be difficult to construct `bytes` of length `0xffffe0` due to upper bound defined by the block gas limit. However, this length is still below Seaport's `OffsetOrLengthMask`, and therefore may be able to evade many checks.

Recommendation: Change the value of `OnlyFullWordMask` to `0xfffffffe0`. But it does not fully fix this problem, as if `len >= 0xfffffffc1`, the calculations can encounter the same issues. But these numbers are impractical in the EVM, and therefore may not be of concern.

There are two potential approaches to this:

1. Revert early if `length` is `>= 0xfffffffc1`. This is the value beyond which the `add(len, 63)` takes more than 32 bits.
2. Or assign a large value for the rounded up length for any values `>= 0xfffffffc1`. `2**32 - 1` is a possibility.

The option 1 would be consistent with Solidity generated code--revert early if length is too large for specific operations.

Here's the above Z3 proof with the extra constraint that length is below `0xfffffffc1`. This is now unsatisfiable.

```
from z3 import *

def AND(x, y):
    return x & y

def ADD(x, y):
    return x + y

n_bits = 256
symb_len = BitVec('Len', n_bits)
const_OnlyFullWordMask = BitVecVal(0xfffffffe0, n_bits)
const_AlmostTwoWords = BitVecVal(0x3f, n_bits)

solver = Solver()

# The expression (for ConsiderationEncoder)
# from https://github.com/ProjectOpenSea/seaport/pull/798/files
expr = AND(ADD(symb_len, const_AlmostTwoWords), const_OnlyFullWordMask)

# Add an upper bound about the length
solver.add(ULT(symb_len, BitVecVal(0xfffffffc1, n_bits)))

# A model where the expression evaluates to a `value < 32`. Such model is now unsatisfiable
solver.add(ULT(expr, BitVecVal(32, n_bits)))

result = solver.check()
if result == sat:
    print("SAT!")
    print(solver.model())
else:
    print(result)
```

Seaport: Partially addressed in PR 798.

Spearbit: Verified.

5.3 Gas Optimization

5.3.1 `_prepareBasicFulfillmentFromCalldata` expands memory more than it's needed by 4 extra words

Severity: Gas Optimization

Context:

- BasicOrderFulfiller.sol#L896

Description: In `_prepareBasicFulfillmentFromCalldata` , we have:

```
// Update the free memory pointer so that event data is persisted.
mstore(0x40, add(0x80, add(eventDataPtr, dataSize)))
```

`OrderFulfilled`'s event data is stored in the memory in the region `[eventDataPtr, eventDataPtr + dataSize)`. It's important to note that `eventDataPtr` is an absolute memory pointer and not a relative one. So the above 4 words, `0x80`, in the snippet are extra.

For example, in the `"ERC721 <=> ETH (basic, minimal and verified on-chain)"` test case in `test/basic.spec.ts` the `Seaport` memory profile at the end of the call of `marketplaceContract.connect(buyer).fulfillBasicOrder(basicOrderParameters, {value,})` looks like:

```
0x000 23b872dd000000000000000000000000f372379f3c48ad9994b46f36f879234a ; transferFrom.selector(from,
↪  to, id)
0x020 27b4556100000000000000000000000016c53175c34f67c1d4dd0878435964c1 ; ...
0x040 0000000000000000000000000000000000000000000000000000000000000440 ; free memory pointer
0x060 0000000000000000000000000000000000000000000000000000000000000000 ; ZERO slot
0x080 fa445660b7e21515a59617fcd68910b487aa5808b8abda3d78bc85df364b2c2f ; orderTypeHash
0x0a0 000000000000000000000000f372379f3c48ad9994b46f36f879234a27b45561 ; offerer
0x0c0 0000000000000000000000000000000000000000000000000000000000000000 ; zone
0x0e0 78d24b64b38e96956003ddebb880ec8c1d01f333f5a4bfba07d65d5c550a3755 ; h(ho)
0x100 81c946a4f4982cb7ed0c258f32da6098760f98eaf6895d9ebbd8f9beccb293e7 ; h(hc, ha[0], ..., ha[n])
0x120 0000000000000000000000000000000000000000000000000000000000000000 ; orderType
0x140 0000000000000000000000000000000000000000000000000000000000000000 ; startTime
0x160 00000000000000000000000000000000ff00000000000000000000000000 ; endTime
0x180 8f1d378d2acd9d4f5883b3b9e85385cf909e7ab825b84f5a6eba28c31ea5246a ; zoneHash > orderHash
0x1a0 0000000000000000000000016c53175c34f67c1d4dd0878435964c1c9b70db7 ; salt > fulfiller
0x1c0 0000000000000000000000000000000000000000000000000000000000000080 ; offererConduitKey > offerer
↪  array head
0x1e0 0000000000000000000000000000000000000000000000000000000000000120 ; counter[offerer] >
↪  consideration array head
0x200 0000000000000000000000000000000000000000000000000000000000000001 ; h[4]? > offer.length
0x220 0000000000000000000000000000000000000000000000000000000000000002 ; h[...]? > offer.itemType
0x240 000000000000000000000000c67947dc8d7fd0c2f25264f9b9313689a4ac39aa ; > offer.token
0x260 00000000000000000000000000000000c02c1411443be3c204092b54976260b9 ; > offer.identifierOrCriteria
0x280 0000000000000000000000000000000000000000000000000000000000000001 ; > offer's current interpolated
↪  amount
0x2a0 0000000000000000000000000000000000000000000000000000000000000001 ; > totalConsiderationRecipients
↪  + 1
0x2c0 0000000000000000000000000000000000000000000000000000000000000000 ; > receivedItemType
0x2e0 0000000000000000000000000000000000000000000000000000000000000000 ; > consideration.token (NATIVE)
0x300 0000000000000000000000000000000000000000000000000000000000000000 ; >
↪  consideration.identifierOrCriteria
0x320 0000000000000000000000000000000000000000000000000000000000000001 ; > consideration's current
↪  interpolated amount
0x340 000000000000000000000000f372379f3c48ad9994b46f36f879234a27b45561 ; > offerer
0x360 0000000000000000000000000000000000000000000000000000000000000000 ; unused
0x380 0000000000000000000000000000000000000000000000000000000000000000 ; unused
0x3a0 0000000000000000000000000000000000000000000000000000000000000000 ; unused
0x3c0 0000000000000000000000000000000000000000000000000000000000000000 ; unused
0x3e0 0000000000000000000000000000000000000000000000000000000000000040 ; sig.length
0x400 26aa4a333d4b615af662e63ce7006883f678068b8dc36f53f70aa79c28f2032c ; sig[ 0:31]
0x420 f640366430611c54bafd13314285f7139c85d69f423794f47ee088fc6bfbf43f ; sig[32:63]
0x440 0000000000000000000000000000000000000000000000000000000000000001 ; fulfilled = 1; // returns
↪  (bool fulfilled)
```

Notice that 4 `unused` memory slots.

- Transaction Trace

This is also a good example to see that certain memory slots that previously held values like `zoneHash`, `salt`, ... have been overwritten to due to the small number of consideration items (this actually happens inside `_prepareBasicFulfillmentFromCalldata`).

Recommendation: The extra `4` memory slot expansion can be removed from the context in question:

```
// Update the free memory pointer so that event data is persisted.
mstore(0x40, add(eventDataPtr, dataSize))
```

Besides reducing the memory expansion this would also save at least 1 `PUSH1` and 1 `ADD`.

Seaport: Fixed in commit e07499.

Spearbit: Verified.

5.3.2 `TypehashDirectory`'s constructor code can be optimized.

Severity: Gas Optimization

Context:

- TypehashDirectory.sol#L75

Description: `TypehashDirectory`'s deployed bytecode in its current form is:

```
00
3ca2711d29384747a8f61d60aad3c450405f7aaff5613541dee28df2d6986d32 ; h_00
bf8e29b89f29ed9b529c154a63038ffca562f8d7cd1e2545dda53a1b582dde30 ; h_01
53c6f6856e13104584dd0797ca2b2779202dc2597c6066a42e0d8fe990b0024d ; h_02
a02eb7ff164c884e5e2c336dc85f81c6a93329d8e9adf214b32729b894de2af1 ; h_03
39c9d33c18e050dda0aeb9a8086fb16fc12d5d64536780e1da7405a800b0b9f6 ; h_04
1c19f71958cdd8f081b4c31f7caf5c010b29d12950be2fa1c95070dc47e30b55 ; h_05
ca74fab2fece9a1d58234a274220ad05ca096a92ef6a1ca1750b9d90c948955c ; h_06
7ff98d9d4e55d876c5cfac10b43c04039522f3ddfb0ea9bfe70c68cfb5c7cc14 ; h_07
bed7be92d41c56f9e59ac7a6272185299b815ddfabc3f25deb51fe55fe2f9e8a ; h_08
d1d97d1ef5eaa37a4ee5fbf234e6f6d64eb511eb562221cd7edfbdde0848da05 ; h_09
896c3f349c4da741c19b37fec49ed2e44d738e775a21d9c9860a69d67a3dae53 ; h_10
bb98d87cc12922b83759626c5f07d72266da9702d19ffad6a514c73a89002f5f ; h_11
e6ae19322608dd1f8a8d56aab48ed9c28be489b689f4b6c91268563efc85f20e ; h_12
6b5b04cbae4fcb1a9d78e7b2dfc51a36933d023cf6e347e03d517b472a852590 ; h_13
d1eb68309202b7106b891e109739dbbd334a1817fe5d6202c939e75cf5e35ca9 ; h_14
1da3eed3ecef6ebaa6e5023c057ec2c75150693fd0dac5c90f4a142f9879fde8 ; h_15
eee9a1392aa395c7002308119a58f2582777a75e54e0c1d5d5437bd2e8bf6222 ; h_16
c3939feff011e53ab8c35ca3370aad54c5df1fc2938cd62543174fa6e7d85877 ; h_17
0efca7572ac20f5ae84db0e2940674f7eca0a4726fa1060ffc2d18cef54b203d ; h_18
5a4f867d3d458dabecad65f6201ceeaba0096df2d0c491cc32e6ea4e64350017 ; h_19
80987079d291feebf21c2230e69add0f283cee0b8be492ca8050b4185a2ff719 ; h_20
3bd8cff538aba49a9c374c806d277181e9651624b3e31111bc0624574f8bca1d ; h_21
5d6a3f098a0bc373f808c619b1bb4028208721b3c4f8d6bc8a874d659814eb76 ; h_22
1d51df90cba8de7637ca3e8fe1e3511d1dc2f23487d05dbdecb781860c21ac1c ; h_23 for height 24
```

Recommendation: It might be cheaper to pre-calculate the hashes off-chain and unroll the loop used since the length is known (24). If this recommendation is accepted, it should be accompanied by unit and differential tests to guarantee the correctness of the hardcoded hash values.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.3.3 `ConsiderationItem.recipient`'s absolute memory offset can be cached and reused

Severity: Gas Optimization

Context:

- OrderCombiner.sol#L388-L391
- OrderCombiner.sol#L402-L405

Description: `ConsiderationItem.recipient`'s absolute offset is calculated twice in the above context.

Recommendation: Perhaps we can cache this calculation and reuse it.

Seaport: Fixed in commit a13bc2.

Spearbit: Verified.

5.3.4 `currentAmount` can potentially be reused when storing this value in memory in `_validateOrdersAndPrepareToFulfill`

Severity: Gas Optimization

Context:

- OrderCombiner.sol#L375
- OrderCombiner.sol#L406-L411

Description: We have

```
considerationItem.startAmount = currentAmount; // 1
...
mload( // 2
    add(
        considerationItem,
        ReceivedItem_amount_offset
    )
)
```

From `1` where `considerationItem.startAmount` is assigned till `2` its value is not modifed.

Recommendation: It might be cheaper to use `currentAmount` (depends on the stack juggling by the compiler):

```
considerationItem.startAmount = currentAmount; // 1
...
currentAmount // 2
```

Seaport: Fixed in PR 828.

Spearbit: Verified.

5.3.5 Information packed in `BasicOrderType` and how `receivedItemType` and `offeredItemType` are derived

Severity: Gas Optimization

Context:

- BasicOrderFulfiller.sol#L142-L145
- BasicOrderFulfiller.sol#L149-L155
- ConsiderationEnums.sol#L23

Description: Currently the way information is packed and unpacked in/from `BasicOrderType` is inefficient. `BasicOrderType` is only used for `BasicOrderParameters` and when unpacking to give an idea how diffferent parameters are packed into this field.

Recommendation: Save gas using bit-packed `uint256` in `BasicOrderParameters` instead:

```
struct BasicOrderParameters {
    ...
    uint256 basicOrderType; // 0x124
    ...
}
```

where the `orderType`, `route`, `receivedItemType` and `offeredItemType` were packed as:

```
0b 00 ... 00 xx yy zz

  zz: orderType
  yy: offeredItemType
  xx: receivedItemType
xxyy: route
```

Then when unpacking these values in `_validateAndFulfillBasicOrder`, we would only need to do:

```
let orderType := and(3, basicOrderType)
let route := and(15, shr(2, basicOrderType))
let offeredItemType := and(3, route)
let receivedItemType := and(3, shr(2, route))
```

And then `BasicOrderRouteType` would also need to be replaced by a collection of allowed route constants:

```
// RECEIVED_TO_OFFERED
uint256 constant ETH_TO_ERC721     = 0x02; // 0b 00 10
uint256 constant ETH_TO_ERC1155    = 0x03; // 0b 00 11
uint256 constant ERC20_TO_ERC721   = 0x06; // 0b 01 10
uint256 constant ERC20_TO_ERC1155  = 0x07; // 0b 01 11
uint256 constant ERC721_TO_ERC20   = 0x09; // 0b 10 01
uint256 constant ERC1155_TO_ERC20  = 0x0d; // 0b 11 01
```

These constants can be used when routing. We can also add additional/final `else` block when `route` is not one of the allowed route constants.

This suggestion would save gas, and simplify/remove the complexity of unpacking `basicOrderType`.

> *Warning*: If backward compatibility for frontend/backend is important for these values, one can apply the following when deriving `receivedItemType` and `offeredItemType`

- `receivedItemType`

Can be simplified to $t = (r \gg 1) + (r > 4)$:

```
receivedItemType := add(shr(1, route), gt(route, 4))
```

or if the codesize has an extra wiggle room:

```
receivedItemType := byte(route, 0x0000010102030000000000000000000000000000000000000000000000000000)
```

- `offeredItemType`

The expression can be simplified to $t =$ `0b011`
$(r \| (1 + (r <$ `0b100`$)))$:

```
offeredItemType := and(3, or(route, add(1, lt(route, 4))))
```

or if the codesize has an extra wiggle room:

```
offeredItemType := byte(route, 0x0203020301010000000000000000000000000000000000000000000000000000)
```

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.3.6 `invalidNativeOfferItemErrorBuffer` calculation can be simplified

Severity: Gas Optimization

Context:

- OrderCombiner.sol#L186-L198

Description: We have:

```
sig                              func
------------------------------------------------------------------------------
0b10101000000101110100010 00 0000100        matchOrders
0b01010101100101000100101 00 1000010        matchAdvancedOrders
0b11101101100110001010010 10 1110100        fulfillAvailableOrders
0b10000111001000000001101 10 1000001        fulfillAvailableAdvancedOrders
              ^ 9th bit
```

Recommendation: The expression for `invalidNativeOfferItemErrorBuffer` can be simplified to save gas:

```
// 231 = 28 * 8 + 7
invalidNativeOfferItemErrorBuffer := and(2, shr(231, calldataload(0)))
```

or even a simpiler (but potentially requires more bytecodes) solution:

```
invalidNativeOfferItemErrorBuffer := and((1 << 232), calldataload(0)))
```

This is dependent on the names and input parameters of these functions and if they get changed, one would need to find a different trick for this calculation.

Seaport: Fixed in PR 864.

Spearbit: Verified.

5.3.7 When accessing or writing to memory the value of an enum for a struct field, the enum's validation is performed

Severity: Gas Optimization

Context:

- CriteriaResolution.sol#L68
- CriteriaResolution.sol#L98
- CriteriaResolution.sol#L150
- CriteriaResolution.sol#L164
- CriteriaResolution.sol#L189
- CriteriaResolution.sol#L215
- Executor.sol#L58
- Executor.sol#L66
- Executor.sol#L81
- FulfillmentApplier.sol#L84
- OrderCombiner.sol#L675
- OrderCombiner.sol#L781
- OrderFulfiller.sol#L206
- OrderFulfiller.sol#L314

- OrderValidator.sol#L134
- OrderValidator.sol#L158
- OrderValidator.sol#L323
- OrderValidator.sol#L593
- ZoneInteraction.sol#L108
- ZoneInteraction.sol#L118

Description: When accessing or writing to memory the value of an `enum` type for a `struct` field, the enum's validation is performed:

```
enum Foo {
   f1,
   f2,
   ...
   fn
}

struct boo {
   Foo foo;
   ...
}

boo memory b;
P(b.foo); // <--- validation will be performed to check whether the value of `b.foo` is out of range
```

This would apply to `OrderComponents.orderType`, `OrderParameters.orderType`, `CriteriaResolver.side`, `OfferItem.itemType`, `ConsiderationItem.itemType`, `SpentItem.itemType`, `ReceivedItem.itemType`, `BasicOrderParameters.basicOrderType`.

Recommendation: If one would like to avoid these validations, `assembly` blocks would need to be used instead of high-level `Solidity`. The `ConsiderationDecoder` currently skips these validations. If validating these values is required, it would be best to consolidate them into one location and also to make sure the validation only happens once.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.3.8 The zero memory slot can be used when supplying no criteria to `fulfillOrder`, `fulfillAvailableOrders`, and `matchOrders`

Severity: Gas Optimization

Context:

- Consideration.sol#L114
- Consideration.sol#L243
- Consideration.sol#L392

Description: When the external functions in this context are called, no criteria is passed to `_validateAndFulfillAdvancedOrder`, `_fulfillAvailableAdvancedOrders`, or `_matchAdvancedOrders`:

```
new CriteriaResolver[](0), // No criteria resolvers supplied.
```

When this gets compiled into `YUL`, the compiler updates the free memory slot by a word and performs an out of range and overflow check for this value:

```
function allocate_memory_<ID>() -> memPtr
{
    memPtr := mload(64)
    let newFreePtr := add(memPtr, 32)
    if or(gt(newFreePtr, 0xffffffffffffffff), lt(newFreePtr, memPtr)) { panic_error_0x41() }
    mstore(64, newFreePtr)
}
```

Recommendation: One can avoid incrementing the free memory pointer and the checks by passing the zero memory slot pointer.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.3.9 `matchOrders`, `matchAdvancedOrders`, `fulfillAvailableAdvancedOrders`, `fulfillAvailableOrders` returns `executions` which is cleaned and validator by the compiler

Severity: Gas Optimization

Context:

- Consideration.sol#L441-L452
- Consideration.sol#L385
- Consideration.sol#L334
- Consideration.sol#L234

Description: Currently, the return values of `matchOrders`, `matchAdvancedOrders`, `fulfillAvailableAdvancedOrders`, `fulfillAvailableOrders` are cleaned and validator by the compiler.

Recommendation: One can use the/a custom encoder to avoid the extra cleanup/validation.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.3.10 `abi.encodePacked` is used when only bytes/string concatenation is needed.

Severity: Gas Optimization

Context:

- ConsiderationBase.sol#L200-L208
- ConsiderationBase.sol#L212-L221
- ConsiderationBase.sol#L225-L239
- ConsiderationBase.sol#L244-L251
- ConsiderationBase.sol#L260-L264
- TypehashDirectory.sol#L27-L35
- TypehashDirectory.sol#L39-L48
- TypehashDirectory.sol#L52-L66
- TypehashDirectory.sol#L94-L100

Description: In the context above, one is using `abi.encodePacked` like the following:

```
bytes memory B = abi.encodePacked(
    "<B1>",
    "<B2>",
    ...
    "<Bn>"
);
```

For each substring, this causes the compiler to use an `mstore` (if the substring occupies more than `32` bytes, it will use the least amount of `mstore`s which is the ceiling of the length of substring divided by `32`), even though multiple substrings can be combined to fill in one memory slot and thus only use `1` `mstore` for those.

Recommendation: It is recommended to convert the above code snippets to:

```
bytes memory B = bytes(
    "<B1>"
    "<B2>"
    ...
    "<Bn>"
);
```

and for the particular case of TypehashDirectory.sol#L94-L100:

```
bytes memory bulkOrderTypeString = abi.encodePacked(
    "BulkOrder(OrderComponents",
    brackets,
    " tree)",
    subTypes
);
```

Replace `abi.encodePacked` with `bytes.concat` or `string.concat` (this rule also applies to ConsiderationBase.sol#L260-L264).

Seaport: Fixed in PR 841.

Spearbit: Verified.

5.3.11 `solc` ABI encoder is used when `OrderFulfilled` is emitted in `_emitOrderFulfilledEvent`

Severity: Gas Optimization

Context: OrderFulfiller.sol#L378-L385

Description: `solc`'s ABI encoder is used when `OrderFulfilled` is emitted in `_emitOrderFulfilledEvent`. That means all the parameters are cleaned and validated before they are provided to `log3`.

Recommendation: We can avoid the clean-up and validation by utilizing the/a custom encoder (`ConsiderationEncoder`).

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.3.12 The use of identity precompile to copy memory need not be optimal across chains

Severity: Gas Optimization

Context: PointerLibraries.sol#L267

Description: The `PointerLibraries` contract uses a `staticcall` to identity precompile, i.e., address `4` to copy memory--poor man's `memcpy`. This is used as a cheaper alternative to copy 32-byte chunks of memory using `mstore(...)` in a for-loop. However, the gas efficiency of the identity precompile relies on the version of the EVM on the underlying chain.

The base call cost for precompiles before Berlin hardfork was 700 (from Tangerine Wistle), and after Berlin, this was reduced to 100 (for warm accounts and precompiles). Many EVM compatible L1s, and even L2s are on old EVM versions. And using the identity precompile would be more expensive than doing `mstores(...)`.

Recommendation: If Seaport is aiming to be optimized across chains, then the tradeoff between `identity` precompile v/s `mstore(...)` needs to be analysed--the overhead of `700` gas may make copies below a certain threshold in words to be more expensive than using `mstore(...)`. However, this optimization need not be worth the maintenance burden.

Seaport: Definitely less important (but still desirable) for it to be fully optimized on other chains. We definitely want the source to be consistent across all chains, however!

Spearbit: Acknowledged.

5.3.13 Use the zero memory slot for allocating empty data

Severity: Gas Optimization

Context: ConsiderationDecoder.sol#L185-L188, ConsiderationDecoder.sol#L416

Description: In cases where an empty data needs to be allocated, one can use the zero slot. This can also be used as initial values for `offer` and `consideration` in `abi_decode_generateOrder_returndata`.

Recommendation: For `getEmptyBytesOrArray`, the following can be made:

```
    function getEmptyBytesOrArray() internal pure returns (MemoryPointer mPtr) {
-       mPtr = malloc(32);
-       mPtr.write(0);
+       mPtr = MemoryPointer.wrap(0x60)
```

In case of `abi_decode_generateOrder_returndata`, if the order is invalid, the function returns a memory pointer pointing to the slot `0x0` for the variables `offer` and `consideration`. The memory region `[0, 32)` is explicitly zeroed to cover this case.

Instead of that, `offer` and `consideration` can be initialized to `0x60`.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.3.14 Some address fields are masked even though the `ConsiderationDecoder` wanted to skip this masking

Severity: Gas Optimization

Context:

- ZoneInteraction.sol#L10
- ZoneInteraction.sol#L61
- OrderValidator.sol#L598
- BasicOrderFulfiller.sol#L269
- BasicOrderFulfiller.sol#L196-L197
- BasicOrderFulfiller.sol#L207
- BasicOrderFulfiller.sol#L222-L223
- BasicOrderFulfiller.sol#L233-L234
- BasicOrderFulfiller.sol#L244
- BasicOrderFulfiller.sol#L246
- BasicOrderFulfiller.sol#L257
- BasicOrderFulfiller.sol#L259
- BasicOrderFulfiller.sol#L269
- OrderCombiner.sol#L562-L563
- OrderCombiner.sol#L591-L592
- OrderCombiner.sol#L458-L459
- OrderFulfiller.sol#L126-L127
- Executor.sol#L65
- Executor.sol#L74
- Executor.sol#L76
- Executor.sol#L84
- Executor.sol#L86
- Executor.sol#L95
- Executor.sol#L97
- Executor.sol#L60 - cleaned `3` times in the same spot
- FulfillmentApplier.sol#L133
- OrderFulfiller.sol#L242
- OrderValidator.sol#L186
- OrderValidator.sol#L367
- ZoneInteraction.sol#L61
- ZoneInteraction.sol#L68
- Consideration.sol#L610
- GettersAndDerivers.sol#L323
- BasicOrderFulfiller.sol#L902

- Consideration.sol#L576
- FulfillmentApplier.sol#L177
- OrderValidator.sol#L634

Subset of

```
find contracts/lib/ -type f -exec grep -nH --color -E
↪  "\.(offerer|zone|token|recipient|considerationToken|offerToken)" {} \;
```

Description: When a field of `address` type from a struct in memory is used, the compiler masks (also: 2, 3) it.

```
struct A {
   address addr;
}

A memory a;

// P is either a statement or a function call
// when compiled --> and(mload(a_addr_pos), 0xffffffffffffffffffffffffffffffffffffffff)
P(a.addr);
```

Also the compiler is making use of

```
function cleanup_address(value) -> cleaned
{
    cleaned := and(value, 0xffffffffffffffffffffffffffffffffffffffff)
}

function abi_encode_address(value, pos)
{
    mstore(pos, and(value, 0xffffffffffffffffffffffffffffffffffffffff))
}
```

in a few places

Recommendation: To prevent this masking (which is also what is intended by the `ConsiderationDecoder` for fields like `zone`, `offerer`, `recipient`, `token` ...), one should instead write statements in assembly and also when passing variables to functions, memory or calldata pointers should be used.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.3.15 `div(x, (1<<n))` can be transformed into `shr(n, x)`

Severity: Gas Optimization

Context:

- LowLevelHelpers.sol#L60-L63
- LowLevelHelpers.sol#L68
- LowLevelHelpers.sol#L79-L85
- TokenTransferrer.sol#L102-L105
- TokenTransferrer.sol#L111
- TokenTransferrer.sol#L128-L137
- TokenTransferrer.sol#L296-L299
- TokenTransferrer.sol#L304

- TokenTransferrer.sol#L318-L324
- TokenTransferrer.sol#L437-L440
- TokenTransferrer.sol#L445
- TokenTransferrer.sol#L459-L465
- TokenTransferrer.sol#L695-L698
- TokenTransferrer.sol#L708
- TokenTransferrer.sol#L722-L731
- Verifiers.sol#L153

Description: The above context, one is dividing a number by a constant which is power of 2:

```
div(x, c) // where c = 1 << n
```

One can perform the same operation using `shr` which cost less gas.

Recommendation: Change the code in the above context to:

```
shr(x, n)
```

`solc` has an optimization rule for this transformation, but depending on which/when optimization steps are used, it would always be best to enforce this rule in the codebase.

The mentioned optimization rule is implemented in this PR 11089.

Seaport: Fixed in PR 858.

Spearbit: Verified.

5.3.16 A note on `pushN`s

Severity: Gas Optimization

Description: We have `17 + 76` `PUSH32`s in the compiled code

```
    17 PUSH32 0x0000000000000000000000000000000000000000000000000000000000000000 // ?? (might be
↪ placeholders for loadimmutable)
    15 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC // used for sub(x,
↪ 4) -> add(x,    -4), used in dispatching to compare calldatasize with another value
     8 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE0 // used for sub(x,
↪ 0x20) -> add(x, -0x20)
     6 PUSH32 0x4CE34AA2000000000000000000000000000000000000000000000000000000000 //
↪ Conduit_execute_signature
     5 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF // used for sub(x,
↪ 1) -> add(x,    -1)
     4 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00 // used to mask out
↪ isValidated
     4 PUSH32 0x4E487B7100000000000000000000000000000000000000000000000000000000 //
↪ Panic_error_selector
     3 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE4 // used for sub(x,
↪ 0x3c - 0x20) -> add(x, -(0x3c - 0x20)), related to accumulator token offset
     3 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC4 // related to
↪ accumulator itemType offset
     3 PUSH32 0x9D9AF8E38D66C62E2C12F0225249FD9D721C54B83F48D9352C97C6CACDCB6F31 // OrderFulfilled
↪ topic0
     2 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE1 // used for sub(x,
↪ 0x1f) -> add(x, -0x1f)
     2 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC0 // used for sub(x,
↪ 0x40) -> add(x, -0x40)
     2 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF0000000000000000000000000000000FF // masks numerator
↪ and isCancelled
     2 PUSH32 0xFFFFFFFF00000000000000000000000000000000000000000000000000000000 // masks Conduit's
↪ return data to compare with ConduitInterface.execute.selector
     2 PUSH32 0x23B872DD00000000000000000000000000000000000000000000000000000000 //
↪ ERC20_transferFrom_signature
     2 PUSH32 0x1901000000000000000000000000000000000000000000000000000000000000 // EIP_712_PREFIX
     1 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE // used for sub(x,
↪ 2) -> add(x,    -2) , when calculating receivedItemType
     1 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFBF // used for sub(x,
↪ 0x41) -> add(x, -0x41), height := div(sub(fullLength, signatureLength), 0x20)
     1 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFBC // used for sub(x,
↪ 0x44) -> add(x, -0x44), EIP1271_isValidSignature_selector_negativeOffset
     1 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF9D // used for sub(x,
↪ 0x63) -> add(x, -0x63), sub(length, BulkOrderProof_minSize)
     1 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEA1 // used for sub(x,
↪ 0x63) -> add(x, -0x63), calldata_array_index_access_struct_OrderComponents_calldata_dyn_calldata
     1 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF0000 // mask used to mask
↪ out isCancelled and isValidated before canceling an order
     1 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF0000000000000000000000000000000000 // used to only mask
↪ the first 15 bytes (denominator) update_storage_value_offsett_uint120_to_t_uint120
     1 PUSH32 0xF280791EFE782EDCF06CE15C8F4DFF17601DB3B88EB3805A0DB7D77FAF757F04 //
↪ OrderValidated(orderHash, orderParameters) event topic0
     1 PUSH32 0xF242432A00000000000000000000000000000000000000000000000000000000 //
↪ ERC1155_safeTransferFrom_signature
     1 PUSH32 0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF //
↪ EIP2098_allButHighestBitMask
     1 PUSH32 0x721C20121297512B72821B97F5326877EA8ECF4BB9948FEA5BFCB6453074D37F //
↪ CounterIncremented(uint256,address) event topic0
     1 PUSH32 0x6BACC01DBE442496068F7D234EDD811F1A5F833243E0AEC824F86AB861F3C90D //
↪ OrderCancelled(bytes32,address,address) event topic 0
     1 PUSH32 0x1626BA7E00000000000000000000000000000000000000000000000000000000 //
↪ EIP1271_isValidSignature_selector
```

stat of `push` instructions excluding `push2`

```
   213 PUSH1 0x0
```

```
  212 PUSH1 0x20
  194 PUSH1 0x40
  111 PUSH1 0x1
   88 PUSH1 0x60
   78 PUSH1 0x4
   77 PUSH1 0x80
   69 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF // when x.addr is passed to a function it
↪  gets cleaned, one can avoid this cleaning procedure
   58 PUSH1 0xA0
   58 PUSH1 0x5
   56 PUSH1 0x1C
   43 PUSH1 0x24
   38 PUSH1 0xC0
   24 PUSH1 0x44
   24 PUSH1 0x3
   23 PUSH4 0xFFFFFFFF
   23 PUSH1 0xE0
   21 PUSH1 0x2
   20 PUSH8 0xFFFFFFFFFFFFFFFF
   17 PUSH32 0x0
   17 PUSH1 0x64
   16 PUSH15 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
   15 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC
   10 PUSH1 0xFF
   10 PUSH1 0xA4
    9 PUSH5 0x1FFFFFFFE0
    9 PUSH1 0xC4
    9 PUSH1 0x6
    9 PUSH1 0x1F
    8 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE0
    8 PUSH1 0x84
    8 PUSH1 0x10
    6 PUSH32 0x4CE34AA2000000000000000000000000000000000000000000000000000000000
    6 PUSH1 0x88
    5 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
    4 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00
    4 PUSH32 0x4E487B7100000000000000000000000000000000000000000000000000000000
    4 PUSH1 0xE4
    4 PUSH1 0x9
    4 PUSH1 0x8
    4 PUSH1 0x22
    3 PUSH4 0xFB5014FC
    3 PUSH4 0xF486BC87
    3 PUSH4 0x5F15D672
    3 PUSH4 0x1A783B8D
    3 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE4
    3 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC4
    3 PUSH32 0x9D9AF8E38D66C62E2C12F0225249FD9D721C54B83F48D9352C97C6CACDCB6F31
    3 PUSH1 0x41
    3 PUSH1 0x11
    2 PUSH4 0xD13D53D4
    2 PUSH4 0x93979285
    2 PUSH4 0x4E487B71
    2 PUSH4 0x375C24C1
    2 PUSH4 0x1A515574
    2 PUSH4 0x17B1F942
    2 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE1
    2 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC0
    2 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF0000000000000000000000000000000FF
    2 PUSH32 0xFFFFFFFF0000000000000000000000000000000000000000000000000000000000
    2 PUSH32 0x23B872DD0000000000000000000000000000000000000000000000000000000000
    2 PUSH32 0x1901000000000000000000000000000000000000000000000000000000000000
```

```
2 PUSH3 0xFFFFE0
2 PUSH21 0xFF0000000000000000000000000000000000000000
2 PUSH17 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFF0000
2 PUSH1 0xB
2 PUSH1 0x7
2 PUSH1 0x55
2 PUSH1 0x45
2 PUSH1 0x42
2 PUSH1 0x3F
2 PUSH1 0x1D
1 PUSH8 0x7536561706F7274
1 PUSH5 0x736F6C6343
1 PUSH5 0x3FFFFFFFC0
1 PUSH5 0x101000000
1 PUSH4 0xFD9F1E10
1 PUSH4 0xFB0F3EE1
1 PUSH4 0xF4DD92CE
1 PUSH4 0xF47B7740
1 PUSH4 0xF07EC373
1 PUSH4 0xEE9E0E63
1 PUSH4 0xED98A574
1 PUSH4 0xE7ACAB24
1 PUSH4 0xD6929332
1 PUSH4 0xD5DA9A1B
1 PUSH4 0xC63CF089
1 PUSH4 0xBFB3F8CE
1 PUSH4 0xBCED929D
1 PUSH4 0xBA832FDD
1 PUSH4 0xB3A34C4C
1 PUSH4 0xA900866B
1 PUSH4 0xA8930E9A
1 PUSH4 0xA8174404
1 PUSH4 0xA61BE9F0
1 PUSH4 0xA5F54208
1 PUSH4 0xA11B63FF
1 PUSH4 0x9BDE339
1 PUSH4 0x98E9DB6E
1 PUSH4 0x98919765
1 PUSH4 0x98891923
1 PUSH4 0x94EB6AF6
1 PUSH4 0x91B3E514
1 PUSH4 0x8BAA579F
1 PUSH4 0x88147732
1 PUSH4 0x87201B41
1 PUSH4 0x815E1D64
1 PUSH4 0x80EC7374
1 PUSH4 0x7FDA7279
1 PUSH4 0x7FA8A987
1 PUSH4 0x79DF72BD
1 PUSH4 0x6FDDE03
1 PUSH4 0x6AB37CE7
1 PUSH4 0x69F95827
1 PUSH4 0x6088D7DE
1 PUSH4 0x5B34B966
1 PUSH4 0x5A052B32
1 PUSH4 0x55944A42
1 PUSH4 0x4F7FB80D
1 PUSH4 0x470C7C1D
1 PUSH4 0x466AA616
1 PUSH4 0x46423AA7
1 PUSH4 0x39F3E3FD
1 PUSH4 0x3312E32
```

```
1 PUSH4 0x21CCFEB7
1 PUSH4 0x1F003D0A
1 PUSH4 0x1CF99B26
1 PUSH4 0x133C37C6
1 PUSH4 0x12D3F5A3
1 PUSH4 0x10FDA3E1
1 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE
1 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFBF
1 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFBC
1 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF9D
1 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEA1
1 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF0000
1 PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFF000000000000000000000000000000000
1 PUSH32 0xF280791EFE782EDCF06CE15C8F4DFF17601DB3B88EB3805A0DB7D77FAF757F04
1 PUSH32 0xF242432A0000000000000000000000000000000000000000000000000000000000
1 PUSH32 0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
1 PUSH32 0x721C20121297512B72821B97F5326877EA8ECF4BB9948FEA5BFCB6453074D37F
1 PUSH32 0x6BACC01DBE442496068F7D234EDD811F1A5F833243E0AEC824F86AB861F3C90D
1 PUSH32 0x1626BA7E0000000000000000000000000000000000000000000000000000000000
1 PUSH18 0x10000000000000000000000000010001
1 PUSH17 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
1 PUSH12 0xFFFFFFFFFFFFFFFFFFFFFFFF
1 PUSH1 0xE8
1 PUSH1 0xE3
1 PUSH1 0x63
1 PUSH1 0x47
1 PUSH1 0x32
1 PUSH1 0x23
1 PUSH1 0x21
1 PUSH1 0x1B
1 PUSH1 0x18
```

- `PUSH32 0x0000000000000000000000000000000000000000000000000000000000000000`

Not sure, why the compiler is including these long `pushN 0`s, they can be replaced to `push1 0`. Filed an issue for it here: Issue 13834

They might correspond to immutables that will be filled at a later stage (`loadimmutable("dddd")`).

- `PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC`

This `push32` is used by the compiler when dispatching to make sure the `calldatasize` is at least the minimum number required by the `external` endpoint.

Refer to for optimization to *use fallback to circumvent solidity's dispatcher mechanism*.

- `PUSH32 0x4CE34AA2000000000000000000000000000000000000000000000000000000000`

This value is used for `Conduit_execute_signature` when populating a memory region to call a conduit:

```
// Write ConduitInterface.execute.selector to memory.
mstore(callDataOffset, Conduit_execute_signature)
```

One can start the `execute` data at the same offset as the errors `0x1c` so that this value is reduced to `PUSH4 0x4CE34AA2`. This might add `3` gas (due to calculating the offset) for some runtime routes which might not be ideal.

- `PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00`
- `PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF0000000000000000000000000000000FF`
- `PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF0000`
- `PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFF000000000000000000000000000000000`

These masks are used when storing `orderStatus` values, the following issue should solve using these masks

- *The arithmetic in _validateOrderAndUpdateStatus can be simplified/optimized*
- PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC
- PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE0
- PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
- PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE4
- PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE1
- PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC0
- PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE
- PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFBF
- PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFBC
- PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF9D
- PUSH32 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEA1

These constants are used when the optimizer transforms `sub(X, C)` into `add(X, -C)` (here `X` is a variable and `C` is a constant). This `Expression Simplifier` optimization step does the transform so that the commutative operation `add` is used instead which allows one to move constants around and potentially combine them. When this optimization rule does not add any benefits and when `C` is a relatively small value, the transformation could grow the bytecode size. To prevent the optimizer steps to perform this transformation, one can use `verbatim` statements in `YUL`:

```
/* let y := sub(x, 0xcc)
 * 60cc | push1 0xcc
 * 90   | swap1
 * 03   | sub
 */
let y := verbatim_1i_1o(hex"60cc_90_03", x)
```

Depending on how `solc` would juggle the stack around, the above suggestion might reduce the code size and might not change the runtime gas (one would need to run a gas diff). The above can be applied to for example when one does:

```
height := div(sub(fullLength, signatureLength), 0x20) // or
sub(length, BulkOrderProof_minSize)
```

- PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

See *Some address fields are masked even though the ConsiderationDecoder wanted to skip this masking*.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.3.17 Use `fallback()` to circumvent Solidity's dispatcher mechanism

Severity: Gas Optimization

Context:

- Consideration.sol#L39

Description: Among other things, the optimization steps are adding extra byte codes that are unnecessary for the dispatching mechanism. For example the `Expression Simplifer` is transforming the following calldata size comparisons:

```
// slt(sub(calldatasize(), 4), X)

push1 0x4
calldatasize
sub
slt
```

into:

```
// slt(add(calldatasize(), 0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffc), X)

push32 0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffc
calldatasize
add
slt
```

And this happens for each exposed endpoint.

This particular optimization rule is helpful if one could reorder and combine the constant value with another one ($A + (X - B) \rightarrow (A - B) + X$, here A, B are constants and X is a variable).

But in this particular instance, the dispatcher does not perform better or worse in regards to the runtime code gas (it stays the same) but the optimization grows the bytecode size.

- Note: The final bytecode depends on the options provided to `solc`. For the above finding, the default `hardhat` settings is used without the `NO_SPECIALIZER` flag.

Recommendation: We can take advantage of the `fallback()` function to avoid the above scenarios. All `external` functions would need to be removed and instead the dispatching would need to happens manually in the `fallback()` function.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.3.18 The arithmetic in `_validateOrderAndUpdateStatus` can be simplified/optimized

Severity: Gas Optimization

Context:

- OrderValidator.sol#L107
- OrderValidator.sol#L192-L291

Description: The arithmetic involving `advancedOrder.numerator`, `advancedOrder.denominator`, `orderStatus.numerator` and `orderStatus.denominator` contains multiple nested `if/else` blocks and for certain conditions/paths extra operations are performed.

variable	description
n_a	`advancedOrder.numerator`

variable	description
d_a	`advancedOrder.denominator`
n_s	`orderStatus.numerator`
d_s	`orderStatus.denominator`

Depending on the case, the final outputs need to be:

- Case 1. $d_s = 0$

In this case n_a, d_a will be unmodified (besides the constraint checks)

$$(n_a, n_s, d_a, d_s) = (n_a, n_a, d_a, d_a)$$

- Case 2. $d_s \neq 0, d_a = 1$

In this case the remaining of the order will be filled and we would have

$$(n_a, n_s, d_a, d_s) = (d_s - n_s, d_s, d_s, d_s)$$

Note that the invariant $d \geq n$ for new fractions and the combined ones is always guaranteed and so $d_s - n_s$ would not underflow.

- Case 3. $d_s \neq 0, d_a \neq 1, d_a = d_s$

Below $\epsilon = (n_a + n_s > d_s)(n_a + n_s - d_s)$ is choosen so that order would not be overfilled. The parameters used in calculating ϵ are taken before they have been updated.

$$(n_a, n_s, d_a, d_s) = (n_a - \epsilon, n_a + n_s - \epsilon, d_s, d_s)$$

- Case 4. $d_s \neq 0, d_a \neq 1, d_a \neq d_s$

Below $\epsilon = (n_a d_s + n_s d_a > d_a d_s)(n_a d_s + n_s d_a - d_a d_s)$ is choosen so that order would not be overfilled. And in case the new values go beyond 120 bits, $G = gcd(n_a d_s - \epsilon, n_a d_s + n_s d_a - \epsilon, d_a d_s)$, otherwise G will be 1. The parameters used in calculating ϵ, G are taken before they have been updated.

$$(n_a, n_s, d_a, d_s) = \frac{1}{G}(n_a d_s - \epsilon, n_a d_s + n_s d_a - \epsilon, d_a d_s, d_a d_s)$$

If one of the updated values occupies more than 120 bits, the call will be reverted.

Recommendation: It would be best to rewrite the logic for when fractions are combined for better readability and also in a more optimized way.

Rough diff draft below.

```
diff --git a/contracts/lib/ConsiderationConstants.sol b/contracts/lib/ConsiderationConstants.sol
index 0879b60d..cde8b9d4 100644
--- a/contracts/lib/ConsiderationConstants.sol
+++ b/contracts/lib/ConsiderationConstants.sol
@@ -102,6 +102,10 @@ uint256 constant AdvancedOrder_denominator_offset = 0x40;
 uint256 constant AdvancedOrder_signature_offset = 0x60;
 uint256 constant AdvancedOrder_extraData_offset = 0x80;

+uint256 constant OrderStatus_ValidatedAndNotCancelled = 1;
+uint256 constant OrderStatus_filledNumerator_offset = 0x10;
+uint256 constant OrderStatus_filledDenominator_offset = 0x88;
+
 uint256 constant AlmostOneWord = 0x1f;
```

```
 uint256 constant OneWord = 0x20;
 uint256 constant TwoWords = 0x40;
diff --git a/contracts/lib/OrderValidator.sol b/contracts/lib/OrderValidator.sol
index f57528b2..f6c373a5 100644
--- a/contracts/lib/OrderValidator.sol
+++ b/contracts/lib/OrderValidator.sol
@@ -100,9 +100,9 @@ contract OrderValidator is Executor, ZoneInteraction {
      *                          order is invalid due to the time or status.
      *
      * @return orderHash      The order hash.
-     * @return newNumerator   A value indicating the portion of the order that
+     * @return numerator      A value indicating the portion of the order that
      *                        will be filled.
-     * @return newDenominator A value indicating the total size of the order.
+     * @return denominator    A value indicating the total size of the order.
      */
     function _validateOrderAndUpdateStatus(
         AdvancedOrder memory advancedOrder,
@@ -111,8 +111,8 @@ contract OrderValidator is Executor, ZoneInteraction {
         internal
         returns (
             bytes32 orderHash,
-            uint256 newNumerator,
-            uint256 newDenominator
+            uint256 numerator,
+            uint256 denominator
         )
     {
         // Retrieve the parameters for the order.
@@ -144,8 +144,18 @@ contract OrderValidator is Executor, ZoneInteraction {
         }

         // Read numerator and denominator from memory and place on the stack.
-        uint256 numerator = uint256(advancedOrder.numerator);
-        uint256 denominator = uint256(advancedOrder.denominator);
+        // Overflowed values would be masked
+        assembly {
+            numerator := and(
+                mload(add(advancedOrder, AdvancedOrder_numerator_offset)),
+                MaxUint120
+            )
+
+            denominator := and(
+                mload(add(advancedOrder, AdvancedOrder_denominator_offset)),
+                MaxUint120
+            )
+        }

         // Ensure that the supplied numerator and denominator are valid.
         if (numerator > denominator || numerator == 0) {
@@ -189,43 +199,85 @@ contract OrderValidator is Executor, ZoneInteraction {
             );
         }

-        // Read filled amount as numerator and denominator and put on the stack.
-        uint256 filledNumerator = orderStatus.numerator;
-        uint256 filledDenominator = orderStatus.denominator;
-
-        // If order (orderStatus) currently has a non-zero denominator it is
-        // partially filled.
-        if (filledDenominator != 0) {
-            // If denominator of 1 supplied, fill all remaining amount on order.
```

```
-            if (denominator == 1) {
-                // Scale numerator & denominator to match current denominator.
-                numerator = filledDenominator;
-                denominator = filledDenominator;
-            }
-            // Otherwise, if supplied denominator differs from current one...
-            else if (filledDenominator != denominator) {
-                // scale current numerator by the supplied denominator, then...
-                filledNumerator *= denominator;
-
-                // the supplied numerator & denominator by current denominator.
-                numerator *= filledDenominator;
-                denominator *= filledDenominator;
-            }
+        assembly {
+            let orderStatusSlot := orderStatus.slot
+            // Read filled amount as numerator and denominator and put on the stack.
+            let filledNumerator := sload(orderStatusSlot)
+            let filledDenominator := shr(
+                OrderStatus_filledDenominator_offset,
+                filledNumerator
+            )

-            // Once adjusted, if current+supplied numerator exceeds denominator:
-            if (filledNumerator + numerator > denominator) {
-                // Skip underflow check: denominator >= orderStatus.numerator
-                unchecked {
-                    // Reduce current numerator so it + supplied = denominator.
-                    numerator = denominator - filledNumerator;
+            for {} 1 {} {
+                if iszero(filledDenominator) {
+                    filledNumerator := numerator
+
+                    break
                 }
-            }

-            // Increment the filled numerator by the new numerator.
-            filledNumerator += numerator;
+                // shift and mask to calculate the the current filled numerator
+                filledNumerator := and(
+                    shr(OrderStatus_filledNumerator_offset, filledNumerator),
+                    MaxUint120
+                )
+
+                // If denominator of 1 supplied, fill all remaining amount on order.
+                if eq(denominator, 1) {
+                    numerator := sub(filledDenominator, filledNumerator)
+                    denominator := filledDenominator
+                    filledNumerator := filledDenominator
+
+                    break
+                }
+
+                // If supplied denominator equals to the current one
+                if eq(denominator, filledDenominator) {
+                    // Increment the filled numerator by the new numerator.
+                    filledNumerator := add(numerator, filledNumerator)
+
+                    // Once adjusted, if current+supplied numerator exceeds denominator:
+                    let _carry := mul(
+                        sub(filledNumerator, denominator),
```

```
+                        gt(filledNumerator, denominator)
+                    )
+
+                    numerator := sub(
+                        numerator,
+                        _carry
+                    )
+
+                    filledNumerator := sub(
+                        filledNumerator,
+                        _carry
+                    )
+
+                    break
+                }
+
+                // Otherwise, if supplied denominator differs from current one...
+                filledNumerator := mul(filledNumerator, denominator)
+                numerator := mul(numerator, filledDenominator)
+                denominator := mul(denominator, filledDenominator)
+
+                // Increment the filled numerator by the new numerator.
+                filledNumerator := add(numerator, filledNumerator)
+
+                // Once adjusted, if current+supplied numerator exceeds denominator:
+                let _carry := mul(
+                    sub(filledNumerator, denominator),
+                    gt(filledNumerator, denominator)
+                )
+
+                numerator := sub(
+                    numerator,
+                    _carry
+                )
+
+                filledNumerator := sub(
+                    filledNumerator,
+                    _carry
+                )

-            // Use assembly to ensure fractional amounts are below max uint120.
-            assembly {
                 // Check filledNumerator and denominator for uint120 overflow.
                 if or(
                     gt(filledNumerator, MaxUint120),
@@ -267,28 +319,25 @@ contract OrderValidator is Executor, ZoneInteraction {
                         // Store the arithmetic (0x11) panic code.
                         mstore(Panic_error_code_ptr, Panic_arithmetic)
                         // revert(abi.encodeWithSignature("Panic(uint256)", 0x11))
-                        revert(0x1c, Panic_error_length)
+                        revert(Error_selector_offset, Panic_error_length)
                     }
                 }
+
+                break
             }
-            // Skip overflow check: checked above unless numerator is reduced.
-            unchecked {
-                // Update order status and fill amount, packing struct values.
-                orderStatus.isValidated = true;
-                orderStatus.isCancelled = false;
-                orderStatus.numerator = uint120(filledNumerator);
```

```
-                orderStatus.denominator = uint120(denominator);
-            }
-        } else {
+
             // Update order status and fill amount, packing struct values.
-            orderStatus.isValidated = true;
-            orderStatus.isCancelled = false;
-            orderStatus.numerator = uint120(numerator);
-            orderStatus.denominator = uint120(denominator);
+            // [denominator: 15 bytes] [numerator: 15 bytes] [isCanecelled: 1 byte] [isValidated: 1
↪  byte]
+            sstore(orderStatusSlot,
+                or(
+                    OrderStatus_ValidatedAndNotCancelled,
+                    or(
+                        shl(OrderStatus_filledNumerator_offset, filledNumerator),
+                        shl(OrderStatus_filledDenominator_offset, denominator)
+                    )
+                )
+            )
         }
-
-        // Return order hash, a modified numerator, and a modified denominator.
-        return (orderHash, numerator, denominator);
     }

     /**
```

Seaport: Fixed in PR 818.

Spearbit: Verified.

5.3.19 Redundant use of `OffsetOrLengthMask`

Severity: Gas Optimization

Context: ConsiderationEncoder.sol#L59

Description:

```
unchecked {
    size =
        ((src.readUint256() & OffsetOrLengthMask) + AlmostTwoWords) &
        OnlyFullWordMask;
    ...
}
```

The mask in `((src.readUint256() & OffsetOrLengthMask)` is redundant, and the following expression will be equivalent:

```
size = (src.readUint256() + AlmostTwoWords) & OnlyFullWordMask;
```

Seaport: Fixed in PR 798.

Spearbit: Verified.

5.4 Informational

5.4.1 Deviations between standard ABI routines and abi-lity

Severity: Informational

Context:

- ConsiderationEncoder.sol
- ConsiderationDecoder.sol

Description:

1. `readMaskedUint256`: masks the higher order bits for calldata pointers. In high level solidity, this would revert.
2. `readBool`, etc: does not check for higher order bits, nor clean the value. In high-level solidity, this would revert.
3. `readInt8`, etc: similar to the `readBool` case. This is interesting because sometimes a `signextend` maybe needed, if the value is used with `sdiv` or `smod`.
4. `abi_encode_bytes` does not clean up data at the end of the last word. For example, consider a `bytes memory` of length `1`. Then the routine copies `64` bytes of data. However, we can only guarantee the integrity of the first `32 + 8 = 40` bytes.
5. `abi_decode_bytes` has the same types of issues as `abi_encode_bytes`.
6. `abi_decode_generateOrder_returndata` expects stricter ABI encoding, i.e., the encoding cannot have any extra data.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.4.2 The magic return value checks can be made stricter

Severity: Informational

Context: ZoneInteraction.sol#L186

Description: The magic return value check for `ZoneInteraction` can be made stricter.

1. It does not check the lower 28 bytes of the return value.
2. It does not check if `extcodesize()` of the `zone` is non-zero. In particular, for the identity precompile, the magic check would pass. This is, however, a general issue with the pattern where magic values are the same as the function selector and not specific to the Zone.

Recommendation: The following snippet would additionally check that the lower 28 bytes of `returndata` is also `0`. This would also save some gas.

```
// The data [4:32) is 0.
let magic := mload(callData)
// Uses a stricter ABI standard, the lower bits are also checked.
magicMatch := eq(magic, mload(0))
```

If we want the magic return value check to fail for the identity precompile, then the `call` should be prefaced by an `extcodesize()` check. This would, however, increase the gas by 100.

Seaport: It's probably a little safer to mask calldata and compare to an unshifted mload to also compare against the 28 lower bits being empty or unreturned.

This was changed in PR 800 to

```
let magic := and(mload(callData), MaskOverFirstFourBytes)
magicMatch := eq(magic, mload(0))
```

Spearbit: Verified.

5.4.3 Resolving additional offer items supplied by contract orders with criteria can be impractical

Severity: Informational

Context: CriteriaResolution.sol#L43

Description: Contract orders can supply additional offer amounts when the order is executed. However, if they supply extra offer items with criteria, on the fly, the fulfiller won't be able to supply the necessary criteria resolvers (the correct Merkle proofs). This can lead to flaky orders that are impractical to fulfill.

Recommendation: Contract offerers should avoid mismatches between `previewOrder` and what's executed on-chain. This can be impractical for complex orders. However, offering additional offer items with criteria is something a contract offerer should rarely do, this should be discouraged. Also see Empty criteriaResolvers for criteria-based contract orders.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.4.4 Use of confusing named constant `SpentItem_size` in a function that deals with only `ReceivedItem`

Severity: Informational

Context: ConsiderationDecoder.sol#L568

Description: The named constant `SpentItem_size` is used in the function `copyReceivedItemsAsConsiderationItems`, even though the context has nothing to do with `SpentItem`.

Recommendation: Consider making a new named constant and replacing the usage.

Seaport: Fixed in commit 3f2312.

Spearbit: Verified.

5.4.5 The ABI-decoding of `generateOrder returndata` does not have sufficient checks to prevent out of bounds `returndata` reads

Severity: Informational

Context: ConsiderationDecoder.sol#L456-L461

Description: There was some attempt to avoid out of bounds `returndata` access in the `ConsiderationDecoder`. However, the two `returndatacopy(...)` in `ConsiderationDecoder.sol#L456-L461` can still lead to out of bounds access and therefore may revert.

Assume that code reaches the line `ConsiderationDecoder.sol#L456`. We have the following constraints

1. `returndatasize >= 4 * 32`: `ConsiderationDecoder.sol#L428`
2. `offsetOffer <= returndatasize`: `ConsiderationDecoder.sol#L444`
3. `offsetConsideration <= returndatasize`: `ConsiderationDecoder.sol#L445`

If we pick a returndata that satisfies 1 and let `offsetOffer == offsetConsideration == returndatasize`, all the constraints are true. But the `returndatacopy` would be revert due to an out-of-bounds read.

Note: High-level Solidity avoids reading from out of bounds returndata. This is usually done by checking if `returndatasize()` is large enough for static data types and always doing `returndatacopy` of the form `returndatacopy(x, 0, returndatasize())`.

Recommendation:

1. If the current behaviour is desired, it's worth documenting that those two `returndatacopy(...)` that can lead to OOB access.

2. If a high-level revert, with additional revert data is desired, just like the other cases with OOB `returndata` access, then consider adding checks similar to

```
-  let invalidOfferOffset := gt(offsetOffer, returndatasize())
+  let invalidOfferOffset := gt(add(offsetOffer, 0x20), returndatasize())
```

A similar check for `offsetOffer` as well. However, as with any ABI offset calculations, any `add(...)` will need to be carefully examined to see if it can overflow.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.4.6 Document that contract orders does not support Seaport-native Dutch or English auctions

Severity: Informational

Context:

- ConsiderationEncoder.sol#L440-L443
- ContractOffererInterface.sol#L7

Description: Seaport supports native Dutch and English auctions by allowing `startPrice` and `endPrice` of offer or consideration item to be different. In this case, the current price is derived by interpolating the two different prices. In case of Contract orders, the low-level ABI encoding for `generateOrder` does not interpolate the price. Instead, it uses the value of `startAmount` and assumes that both the values are the same.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.4.7 Consider renaming `writeBytes` to `writeBytes32`

Severity: Informational

Context: PointerLibraries.sol#L3064

Description: The function name `writeBytes` is not accurate in this context.

Recommendation:

```
-    function writeBytes(MemoryPointer mPtr, bytes32 value) internal pure {
+    function writeBytes32(MemoryPointer mPtr, bytes32 value) internal pure {
        assembly {
            mstore(mPtr, value)
        }
    }
```

Seaport: Fixed in commit 25af190.

Spearbit: Verified.

5.4.8 Zones no longer have access to any criteria information

Severity: Informational

Context:

- OrderFulfiller.sol#L117
- OrderFulfiller.sol#L98

Description: The zone interface was changed from `isValidOrderIncludingExtraData` to a generic `validateOrder`, this does not give zones access to the criteria resolvers. Moreover, there is a subtlety introduced by changing the restricted order check, from inside `_validateOrderAndUpdateStatus` to post execution. Any criteria-based order information would be resolved and replaced by `_applyCriteriaResolvers`, this does not give zones access to the original order.

Any zones that have specialized checks based on criteria items will have issues upgrading to work with Seaport 1.2.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.4.9 Missing test case for criteria-based contract orders and `identifierOrCriteria != 0` case

Severity: Informational

Context: advanced.spec.ts#L434

Description: The only test case for criteria-based contract orders in advanced.spec.ts#L434. This tests the case for `identifierOrCriteria == 0`. For the other case, `identifierOrCriteria != 0` tests are missing.

Recommendation: Write tests for criteria-based contract orders and `identifierOrCriteria != 0` case, where a Merkle proof for criteria needs to be provided. Also see *Empty criteriaResolvers for criteria-based contract orders*.

Seaport: Fixed in PR 847 and commit c4da5e5.

Spearbit: Verified.

5.4.10 NatSpec comment for `conduitKey` in `bulkTransfer()` says "optional" instead of "mandatory"

Severity: Informational

Context:

- TransferHelper.sol#L65
- TransferHelper.sol#L75-L77

Description: The NatSpec comment says that `conduitKey` is optional but there is a check making sure that this value is always supplied.

Recommendation:

```
File: TransferHelper.sol
- 65:      * @param conduitKey An optional conduit key referring to a conduit through
+ 65:      * @param conduitKey A mandatory conduit key referring to a conduit through
66:      *                   which the bulk transfer should occur.
...
74:         // Ensure that a conduit key has been supplied.
75:         if (conduitKey == bytes32(0)) {
76:             revert InvalidConduit(conduitKey, address(0));
77:         }
```

Seaport: Fixed in PR 819.

Spearbit: Verified.

5.4.11 As the `_counters` are incremented by `quasiRandomNumber`, it would be hard to sign orders that can only be used when the counter is updated

Severity: Informational

Context:

- CounterManager.sol#L46-L50
- CounterManager.sol#L35

Recommendation: As the `_counters` are incremented by `quasiRandomNumber`, it would be hard to sign orders that can only be used when the counter is updated (since one can't predict the future block hashes):

```
let quasiRandomNumber := shr(128, blockhash(sub(number(), 1)))
```

Seaport: Fixed in PR 837.

Spearbit: Verified.

5.4.12 Comparing the magic values returned by different contracts are inconsistent

Severity: Informational

Context:

- ZoneInteraction.sol#L186
- SignatureVerification.sol#L212

Description: In `ZoneInteraction`'s `_callAndCheckStatus` we perform the following comparison for the returned magic value:

```
let magic := shr(224, mload(callData))
magicMatch := eq(magic, shr(224, mload(0)))
```

But the returned magic value comparison in _assertValidSignature without truncating the returned value:

```
if iszero(eq(mload(0), EIP1271_isValidSignature_selector))
```

Recommendation: It would be best to have a consistent comparison when checking the magic value. Perhaps checking the lower bytes are clean as in `_assertValidSignature` would be the better choice.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.4.13 Document the structure of the `TypehashDirectory`

Severity: Informational

Context:

- TypehashDirectory.sol#L119-L120

Description: Instances of `TypehashDirectory` would act as storage contracts with runtime bytecode:

```
[0x00 - 0x00] 00
[0x01 - 0x20] h(struct BulkOrder { OrderComponents[2]          tree })
[0x21 - 0x40] h(struct BulkOrder { OrderComponents[2][2]       tree })
...
[0xNN - 0xMM] h(struct BulkOrder { OrderComponents[2][2]...[2] tree })
```

`h` calculates the `eip-712` `typeHash` of the input `struct`. `0xMM` would be `mul(MaxTreeHeight, 0x20)` and `0xNN = 0xMM - 0x1f`.

Recommendation: Document the above structure.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.4.14 Document what `twoSubstring` encodes

Severity: Informational

Context:

- TypehashDirectory.sol#L4

Description: We have:

```
bytes32 constant twoSubstring = 0x5B325D00000000000000000000000000000000000000000000000000000000000000;
```

which encodes:

```
cast --to-ascii 0x5B325D00000000000000000000000000000000000000000000000000000000000000
[2]
```

Recommendation: Comment that `0x5B325D00000000000000000000000000000000000000000000000000000000000000` encodes `[2]`.

Seaport: Fixed in PR 799.

Spearbit: Verified.

5.4.15 Upper bits of the `to` parameter to call opcodes are stripped out by clients

Severity: Informational

Context:

- Executor.sol#L238
- OrderValidator.sol#L374
- SignatureVerification.sol#L196
- TokenTransferrer.sol#L59
- TokenTransferrer.sol#L277
- TokenTransferrer.sol#L418
- TokenTransferrer.sol#L676
- ZoneInteraction.sol#L184

Description: Upper bits of the `to` parameter to call opcodes are stripped out by clients. For example, `geth` would strip the upper bytes out:

- instructions.go#L674
- uint256.go#L114-L121

So even though the `to` parameters in this context can have dirty upper bits, the call opcodes can be successful, and masking these values in the contracts is not necessary for this context.

Recommendation: Document this fact.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.4.16 Remove unused functions

Severity: Informational

Context:

- LowLevelHelpers.sol#L25
- LowLevelHelpers.sol#L112
- ZoneInteraction.sol#L227
- PointerLibraries.sol#L3079
- PointerLibraries.sol#L37
- PointerLibraries.sol#L190

Description: The functions in the above context are not used in the codebase.

Recommendation: It you are not planning to use these functions, it would be best to remove them from the codebase.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.4.17 `Fulfillment_itemIndex_offset` can be used instead of `OneWord`

Severity: Informational

Context:

- FulfillmentApplier.sol#L392
- FulfillmentApplier.sol#L678

Description: In the above context, one has:

```
// Get the item index using the fulfillment pointer.
itemIndex := mload(add(mload(fulfillmentHeadPtr), OneWord))
```

Recommendation: `Fulfillment_itemIndex_offset` is already a defined named constant which has the same value as `OneWord` (`0x20`). It would be best to use that constant in this context, since its name is more specific to its usage:

```
itemIndex := mload(add(mload(fulfillmentHeadPtr), Fulfillment_itemIndex_offset))
```

Seaport: Fixed in PR 819.

Spearbit: Verified.

5.4.18 Document how the `_pauser` role is assigned for `PausableZoneController`

Severity: Informational

Context:

- PausableZoneController.sol#L123-L131
- PausableZone.sol#L106-L112

Description: The `_pauser` role is an important role for a `PausableZoneController`. It can pause any `zone` created by this controller and thus transfer all the native token funds locked in that `zone` to itself.

Recommendation: It would be best to document how the `_pauser` is picked for a controller. Currently, the `PausableZoneController` used by OpenSea has an `owner` (4/5 GnosisSafe) and the `_pauser` is also a 2/5 GnosisSafe with the same set of owners.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.4.19 `_aggregateValidFulfillmentConsiderationItems`'s memory layout assumptions depend on `_validateOrdersAndPrepareToFulfill`'s memory manipulation

Severity: Informational

Context:

- FulfillmentApplier.sol#L621-L625
- FulfillmentApplier.sol#L724-L735
- OrderCombiner.sol#L382-L393

Description: In `_aggregateValidFulfillmentConsiderationItems` we are using the `ReceivedItem.recipient`'s offset of `considerationItemPtr` to write to `receivedItem` at the same offset (the same offset is also used here):

```
// Set the recipient on the received item.
mstore(
    add(receivedItem, ReceivedItem_recipient_offset),
    mload(add(considerationItemPtr, ReceivedItem_recipient_offset))
)
```

This looks buggy, but in `_validateOrdersAndPrepareToFulfill(...)` we overwrite `consideration[i].endAmount` with `consideration[i].recipient`:

```
mstore(
    add(
        considerationItem,
        ReceivedItem_recipient_offset // old endAmount
    ),
    mload(
        add(
            considerationItem,
            ConsiderationItem_recipient_offset
        )
    )
)
```

Also `_validateOrdersAndPrepareToFulfill` gets called first in `_fulfillAvailableAdvancedOrders` and `_matchAdvancedOrders`. This is important since the memory for the `consideration` arrays needs to be updated before we reach `_aggregateValidFulfillmentConsiderationItems`.

Recommendation: This observation needs to be documented/commented on both `_validateOrdersAndPrepareToFulfill` and `_aggregateValidFulfillmentConsiderationItems` to emphasis their dependency on memory layout.

Seaport: Fixed in PR 868.

Spearbit: Verified.

5.4.20 `recipient` is provided as the `fulfiller` for the `OrderFulfilled` event

Severity: Informational

Context:

- OrderCombiner.sol#L460
- OrderFulfiller.sol#L12
- OrderFulfiller.sol#L361

Description: In the above context in general it is not true that the `recipient` is the `fulfiller`. Also note that `recipient` is `address(0)` for match orders.

Recommendation: Either the correct parameter needs to be provided to `_emitOrderFulfilledEvent` or the parameter name and comments need to be updated.

Seaport: Fixed in commit 119b1e4.

Spearbit: Verified.

5.4.21 `availableOrders[i]` return values need to be explicitly assigned since they live in a region of memory which might have been dirtied before

Severity: Informational

Context:

- OrderCombiner.sol#L728

Description: Seaport `1.1` did not have the following default assignment:

```
if (advancedOrder.numerator == 0) {
    availableOrders[i] = false;
    continue;
}
```

But this is needed here since the current memory region which was previously used by the `accumulator` might be dirty.

Recommendation: Add a comment to emphasize the above point.

Seaport: Fixed in PR 855.

Spearbit: Verified.

5.4.22 Usage of `MemoryPointer` / formatting inconsistent in `_getGeneratedOrder`

Severity: Informational

Context:

- OrderValidator.sol#L451-L465
- OrderValidator.sol#L416-L427

Description: Usage of `MemoryPointer` / formatting is inconsistent between the loop used `OfferItem`s and the loop used for `ConsiderationItem`s.

Recommendation: It would be best to have a more unified formatting between these 2 loops.

Seaport: Fixed in PR 824.

Spearbit: Verified.

5.4.23 `newAmount` is not used in `_compareItems`

Severity: Informational

Context:

- OrderValidator.sol#L319

Description: `newAmount` is unused in `_compareItems`.

If `originalItem` points to $I = (t, T, i, a_s, a_e)$ and the `newItem` to $I_{new} = (t', T', i', a'_s, a'_e)$ where

parameter	description
t'	`itemType`
t	`itemType` for I after the adjustment for restricted collection items
T, T'	`token`
i'	`identifierOrCriteria`
i	`identifierOrCriteria` for I after the adjustment for restricted collection items
a_s, a'_s	`startAmount`
a_e, a'_e	`endAmount`
c	`_compareItems`

then we have

$$c(I, I_{new}) = (t \neq t') \vee (T \neq T') \vee (i \neq i') \vee (a_s \neq a_e)$$

and so we are not comparing either a_s to a'_s or a'_s to a'_e. In `abi_decode_generateOrder_returndata` $a'_s = a'_e$ is enforced.

In `_getGeneratedOrder` we have the following check: $a_s > a'_s$ (invalid case for offer items that contributes to `errorBuffer`. inequality is reversed for consideration items).

And so in each loop $(t \neq t') \vee (T \neq T') \vee (i \neq i') \vee (a_s \neq a_e) \vee (a_s > a'_s)$ is `or`ed to `errorBuffer`.

Recommendation: If the `newAmount` is not planned to be used in `_compareItems` it can be removed. Document the other constraints on the returned data from the contract `offerer` mentioned above.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.4.24 reformat `validate` so that its body is consistent with the other `external` functions

Severity: Informational

Context:

- Consideration.sol#L488

Description / Recommendation: For consistency with other functions we can rewrite `validate` as:

```
function validate(
    Order[] calldata /* orders */
) external override returns (bool /* validated */) {
    return
        _validate(to_dyn_array_Order_ReturnType(
            abi_decode_dyn_array_Order
        )(CalldataStart.pptr()));
}
```

Needs to be checked if it changes code size or gas cost.

Seaport: Fixed in PR 824.

Spearbit: Verified.

5.4.25 Add commented parameter names (`Type Location /* name */`)

Severity: Informational

Context:

- Consideration.sol#L489

Description / Recommendation: Add commented parameter names (`Type Location /* name */`) for `validate`:

```
Order[] calldata /* orders */
```

Seaport: Fixed in commit 74de34.

Spearbit: Verified.

5.4.26 Document that the `height` provided to `_lookupBulkOrderTypehash` can only be in a certain range

Severity: Informational

Context:

- ConsiderationBase.sol#L270

Description: Need to have `height` h provided to `_lookupBulkOrderTypehash` such that:

$$1 + 32(h - 1) \in [0, \min(\texttt{0xffffffff\_ffffffff}, \text{typeDirectory.codesize}) - 32]$$

Otherwise `typeHash := mload(0)` would be `0` or would be padded by zeros.

When `extcodecopy` gets executed

```
extcodecopy(directory, 0, typeHashOffset, 0x20)
```

clients like `geth` clamp `typehashOffset` to minimum of `0xffffffff_ffffffff` and `directory.codesize` and pads the result with `0`s if out of range.

ref:

- instructions.go#L373

- common.go#L54

Recommendation: Comment on the above limitations/constraints in the code base for `_lookupBulkOrderTypehash`.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.4.27 Unused imports can be removed

Severity: Informational

Context:

- ZoneInterface.sol#L5-L8
- TransferHelperInterface.sol#L5

Description: The imported contents in this context are unused.

Recommendation: The unused imports can be removed.

Seaport: Fixed in PR 829.

Spearbit: Verified.

5.4.28 `msg.sender` is provided as the `fulfiller` input parameter in a few locations

Severity: Informational

Context:

- ConsiderationEncoder.sol#L234
- ConsiderationEncoder.sol#L81-L82

Description: `msg.sender` is provided as the `fulfiller` input parameter.

Recommendation: Either this needs to be corrected or previous documentation/comments regarding what a `fulfiller` means in the context of contract orders needs to be updated.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.4.29 Differences and similarities of `ConsiderationDecoder` and solc‘ when decoding dynamic arrays of static/fixed base struct type

Severity: Informational

Context:

- ConsiderationDecoder.sol#L86
- ConsiderationDecoder.sol#L94

Description: The way `OfferItem[]` in `abi_decode_dyn_array_OfferItem` and `ConsiderationItem[]` in `abi_decode_dyn_array_ConsiderationItem` are decoded are consistent with `solc` regarding this:

- For dynamic arrays of static/fixed base struct type, the memory region looks like:

```
[mPtrLength       : mPtrLength + 0x20)           arrLength
--------------------------------------------------
[mPtrLength + 0x20: mPtrLength + 0x40)           memberTail1 - a memory pointer to the array's 1st
↪  element
...
[mPtrLength +  ...: mPtrLength +  ...)           memberTailN - a memory pointer to the array's Nth
↪  element
--------------------------------------------------
[memberTail1      : memberTail1 + <STRUCT_SIZE>) element1
...
[memberTailN      : memberTailN + <STRUCT_SIZE>) elementN
```

The difference is `solc` decodes and validates (checking dirty bytes) each field of the elements of the array (which are static struct types) separately (one `calldataload` and validation per field per element). `ConsiderationDecoder` skips all those validations for both `OfferItems[]` and `ConsiderationItems[]` by copying a chunk of `calldata` to memory (the tail parts):

```
calldatacopy(
    mPtrTail,
    add(cdPtrLength, 0x20),
    mul(arrLength, OfferItem_size)
)
```

That means for `OfferItem[]`, `itemType` and `token` (and also `recipient` for `ConsiderationItem[]`) fields can potentially have dirty bytes.

Recommendation: Document that struct field validations are skipped and `calldata` is copied to `memory` in chunks.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.4.30 `PointerLibraries`'s `malloc` skips some checks

Severity: Informational

Context:

- PointerLibraries.sol#L26-L31

Description: `malloc` in `PointerLibraries` skips checking if `add(mPtr, size)` is OOR or wraps around. Solidity does the following when allocating memory:

```
function allocate_memory(size) -> memPtr {
  memPtr := allocate_unbounded()
  finalize_allocation(memPtr, size)
}

function allocate_unbounded() -> memPtr {
  memPtr := mload(<freeMemoryPointer>)
}

function finalize_allocation(memPtr, size) {
  let newFreePtr := add(memPtr, round_up_to_mul_of_32(size))
  // protect against overflow
  if or(gt(newFreePtr, 0xffffffff_ffffffff), lt(newFreePtr, memPtr)) { // <-- the check that is skipped
    panic_error_<code>()
  }
  mstore(<freeMemoryPointer>, newFreePtr)
}

function round_up_to_mul_of_32(value) -> result {
  result := and(add(value, 31), not(31))
}

function panic_error_<code>() {
  // <selector> = cast sig "Panic(uint256)"
  mstore(0, <selector>)
  mstore(4, <code>)
  revert(0, 0x24)
}
```

Also note, rounding up the `size` to the nearest word boundary is hoisted out of `malloc`.

Recommendation: The above needs to be documented, especially if this file gets used by other projects. As `Seaport` enforces bounds on `size` outside this library file.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.4.31 `abi_decode_bytes` can populate memory with dirty bytes

Severity: Informational

Context:

- ConsiderationDecoder.sol#L60

Description: When `abi_decode_bytes` decodes `bytes`, it rounds its `size` and copies the rounded size from `calldata` to `memory`. This memory region might get populated with dirty bytes.

So for example:

For both `signature` and `extraData` we are using `abi_decode_bytes`. If the `AdvancedOrder` is tightly packed and:

- If `signature`'s length is not a multiple of a word (`0x20`) part of the `extraData.length` bytes will be copied/duplicated to the end of `signature`'s last memory slot.
- If `extraData`'s length is not a multiple of a word (`0x20`) part of the `calldata` that comes after `extraData`'s tail will be copied to memory.

Even if `AdvancedOrder` is not tightly packed (tail offsets are multiple of a word relative to the head), the user can stuff the `calldata` with dirty bits when `signature`'s or `extraData`'s length is not a multiple of a word. And those dirty bits will be carried over to memory during decoding. Note, these extra bits will not be overridden or

cleaned during the decoding because of the way we use and update the free memory pointer (incremented by the rounded-up number to a multiple of a word).

Recommendation: Care needs to be taken when using the copied memory region if the `size` gets rounded to a greater number to make sure the dirty bytes don't get used.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.4.32 `abi_encode_validateOrder` reuses a memory region

Severity: Informational

Context:

- ConsiderationEncoder.sol#L361-L364

Description: It is really important to note that before `abi_encode_validateOrder` is called, `_prepareBasicFulfillmentFromCalldata(...)` needs to be called to populate the memory region that is used for `event OrderFulfilled(...)` which can be reused/copied in this function:

```
MemoryPointer.wrap(offerDataOffset).copy(
    dstHead.offset(tailOffset),
    offerAndConsiderationSize
);
```

From when the memory region for `OrderFulfilled(...)` is populated till we reach this point, care needs to be taken to not modified that region.

`accumulator` data is written to the memory after that region and the current implementation does not touch that region during the whole call after the event has been emitted.

Recommendation: It is important to document this reuse of memory region and dependence of `abi_encode_validateOrder` and `_prepareBasicFulfillmentFromCalldata(...)`.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.4.33 `abi_encode_validateOrder` writes to a memory region that might have been potentially dirtied by `accumulator`

Severity: Informational

Context:

- ConsiderationEncoder.sol#L210

Description: In `abi_encode_validateOrder` potentially (in the future), we might be writing in an area where `accumulator` was used. And since the book-keeping for the `accumulator` does not update the free memory pointer, we need to make sure all bytes in the memory in the range `[dst, dst+size)` are fully updated/written to in this function.

Recommendation: Leave note and document the above.

Seaport: Fixed in PR 866.

Spearbit: Verified.

5.4.34 Reorder writing to memory in `ConsiderationEncoder` to follow the order in `struct` definitions.

Severity: Informational

Context:

- ConsiderationEncoder.sol#L334-L335

Description: Reorder the memory writes in `ConsiderationEncoder` to follow the order in `struct` definitions.

Recommendation: Change the order of

```
dstHead.offset(ZoneParameters_offerer_offset).write(parameters.offerer);
dstHead.offset(ZoneParameters_fulfiller_offset).write(msg.sender);
```

to

```
dstHead.offset(ZoneParameters_fulfiller_offset).write(msg.sender);
dstHead.offset(ZoneParameters_offerer_offset).write(parameters.offerer);
```

Seaport: Fixed in PR 830.

Spearbit: Verified.

5.4.35 The compiled `YUL` code includes redundant consecutive validation of enum types

Severity: Informational

Description: Half the location where an `enum` type struct field has been used/accessed, the validation function for this `enum` type is performed twice:

```
validator_assert_enum_<ENUM_NAME>(memPtr)
validator_assert_enum_<ENUM_NAME>(memPtr)
```

Recommendation: This is possibly a compiler bug.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.4.36 Consider writing tests for revert functions in `ConsiderationErrors`

Severity: Informational

Context: ConsiderationErrors.sol

Description: `ConsiderationErrors.sol` is a new file and is untested. Writing test cases to make sure the revert functions are throwing the right errors is an easy way to prevent mistakes.

Recommendation: Consider adding the following foundry test file: `ConsiderationErrorsTest.t.sol`

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

import "forge-std/Test.sol";

import "../../contracts/lib/ConsiderationErrors.sol";

import { ConsiderationEventsAndErrors } from
↪  "../../contracts/interfaces/ConsiderationEventsAndErrors.sol";

import { ReentrancyErrors } from "../../contracts/interfaces/ReentrancyErrors.sol";

import { CriteriaResolutionErrors } from "../../contracts/interfaces/CriteriaResolutionErrors.sol";
```

```
import { TokenTransferrerErrors } from "../../contracts/interfaces/TokenTransferrerErrors.sol";

import { ZoneInteractionErrors } from "../../contracts/interfaces/ZoneInteractionErrors.sol";

import { FulfillmentApplicationErrors } from
↪  "../../contracts/interfaces/FulfillmentApplicationErrors.sol";

contract ConsiderationErrorsTests is Test {
    function test__revertInsufficientEtherSupplied() public {
        vm.expectRevert(ConsiderationEventsAndErrors.InsufficientEtherSupplied.selector);
        _revertInsufficientEtherSupplied();
    }

    function test__revertOrderAlreadyFilled(bytes32 orderHash) public {

↪   vm.expectRevert(abi.encodeWithSelector(ConsiderationEventsAndErrors.OrderAlreadyFilled.selector,
↪   orderHash));
        _revertOrderAlreadyFilled(orderHash);
    }

    function test__revertBadFraction() public {
        vm.expectRevert(ConsiderationEventsAndErrors.BadFraction.selector);
        _revertBadFraction();
    }

    function test__revertConsiderationNotMet(uint256 i, uint256 j, uint256 unmetAmount) public {

↪   vm.expectRevert(abi.encodeWithSelector(ConsiderationEventsAndErrors.ConsiderationNotMet.selector,
↪   i, j, unmetAmount));
        _revertConsiderationNotMet(i, j, unmetAmount);
    }

    function test__revertInvalidBasicOrderParameterEncoding() public {
        vm.expectRevert(ConsiderationEventsAndErrors.InvalidBasicOrderParameterEncoding.selector);
        _revertInvalidBasicOrderParameterEncoding();
    }

    function test__revertInvalidCallToConduit(address conduit) public {

↪   vm.expectRevert(abi.encodeWithSelector(ConsiderationEventsAndErrors.InvalidCallToConduit.selector,
↪   conduit));
        _revertInvalidCallToConduit(conduit);
    }

    function test__revertInvalidCanceller() public {
        vm.expectRevert(ConsiderationEventsAndErrors.InvalidCanceller.selector);
        _revertInvalidCanceller();
    }

    function test__revertInvalidConduit(bytes32 conduitKey, address conduit) public {
        vm.expectRevert(abi.encodeWithSelector(ConsiderationEventsAndErrors.InvalidConduit.selector,
↪   conduitKey, conduit));
        _revertInvalidConduit(conduitKey, conduit);
    }

    function test__revertInvalidMsgValue(uint256 value) public {
        vm.expectRevert(abi.encodeWithSelector(ConsiderationEventsAndErrors.InvalidMsgValue.selector,
↪   value));
        _revertInvalidMsgValue(value);
    }
```

```
    function test__revertInvalidNativeOfferItem() public {
        vm.expectRevert(ConsiderationEventsAndErrors.InvalidNativeOfferItem.selector);
        _revertInvalidNativeOfferItem();
    }

    function test__revertInvalidTime(uint256 startTime, uint256 endTime) public {
        vm.expectRevert(abi.encodeWithSelector(ConsiderationEventsAndErrors.InvalidTime.selector,
↪   startTime, endTime));
        _revertInvalidTime(startTime, endTime);
    }

    function test__revertMissingOriginalConsiderationItems() public {
        vm.expectRevert(ConsiderationEventsAndErrors.MissingOriginalConsiderationItems.selector);
        _revertMissingOriginalConsiderationItems();
    }

    function test__revertNoSpecifiedOrdersAvailable() public {
        vm.expectRevert(ConsiderationEventsAndErrors.NoSpecifiedOrdersAvailable.selector);
        _revertNoSpecifiedOrdersAvailable();
    }

    function test__revertOrderIsCancelled(bytes32 orderHash) public {
        vm.expectRevert(abi.encodeWithSelector(ConsiderationEventsAndErrors.OrderIsCancelled.selector,
↪   orderHash));
        _revertOrderIsCancelled(orderHash);
    }

    function test__revertOrderPartiallyFilled(bytes32 orderHash) public {

↪   vm.expectRevert(abi.encodeWithSelector(ConsiderationEventsAndErrors.OrderPartiallyFilled.selector,
↪   orderHash));
        _revertOrderPartiallyFilled(orderHash);
    }

    function test__revertPartialFillsNotEnabledForOrder() public {
        vm.expectRevert(ConsiderationEventsAndErrors.PartialFillsNotEnabledForOrder.selector);
        _revertPartialFillsNotEnabledForOrder();
    }

    function test__revertConsiderationLengthExceedsTotalOriginal() public {
        vm.expectRevert(ConsiderationEventsAndErrors.ConsiderationLengthExceedsTotalOriginal.selector);
        _revertConsiderationLengthExceedsTotalOriginal();
    }

    function test__revertNoReentrantCalls() public {
        vm.expectRevert(ReentrancyErrors.NoReentrantCalls.selector);
        _revertNoReentrantCalls();
    }

    function test__revertCriteriaNotEnabledForItem() public {
        vm.expectRevert(CriteriaResolutionErrors.CriteriaNotEnabledForItem.selector);
        _revertCriteriaNotEnabledForItem();
    }

    function test__revertUnresolvedConsiderationCriteria(uint256 i, uint256 j) public {
        vm.expectRevert(abi.encodeWithSelector(CriteriaResolutionErrors.UnresolvedConsiderationCriteria
↪   .selector, i,
↪   j));
        _revertUnresolvedConsiderationCriteria(i, j);
    }

    function test__revertUnresolvedOfferCriteria(uint256 i, uint256 j) public {
```

```
↪ vm.expectRevert(abi.encodeWithSelector(CriteriaResolutionErrors.UnresolvedOfferCriteria.selector,
↪ i, j));
        _revertUnresolvedOfferCriteria(i, j);
    }

    function test__revertOrderCriteriaResolverOutOfRange() public {
        vm.expectRevert(abi.encodeWithSelector(CriteriaResolutionErrors.OrderCriteriaResolverOutOfRange
↪ .selector,
↪ Side.CONSIDERATION));
        _revertOrderCriteriaResolverOutOfRange(Side.CONSIDERATION);
    }

    function test__revertInvalidProof() public {
        vm.expectRevert(CriteriaResolutionErrors.InvalidProof.selector);
        _revertInvalidProof();
    }

    function test__revertUnusedItemParameters() public {
        vm.expectRevert(TokenTransferrerErrors.UnusedItemParameters.selector);
        _revertUnusedItemParameters();
    }

    function test__revertInvalidERC721TransferAmount(uint256 amount) public {

↪ vm.expectRevert(abi.encodeWithSelector(TokenTransferrerErrors.InvalidERC721TransferAmount.selector,
↪ amount));
        _revertInvalidERC721TransferAmount(amount);
    }

    function test__revertInvalidRestrictedOrder(bytes32 orderHash) public {
        vm.expectRevert(abi.encodeWithSelector(ZoneInteractionErrors.InvalidRestrictedOrder.selector,
↪ orderHash));
        _revertInvalidRestrictedOrder(orderHash);
    }

    function test__revertInvalidContractOrder(bytes32 orderHash) public {
        vm.expectRevert(abi.encodeWithSelector(ZoneInteractionErrors.InvalidContractOrder.selector,
↪ orderHash));
        _revertInvalidContractOrder(orderHash);
    }

    function test__revertMismatchedFulfillmentOfferAndConsiderationComponents(uint256 fulfillmentIndex)
↪ public {
        vm.expectRevert(abi.encodeWithSelector(FulfillmentApplicationErrors.MismatchedFulfillmentOfferA
↪ ndConsiderationComponents.selector,
↪ fulfillmentIndex));
        _revertMismatchedFulfillmentOfferAndConsiderationComponents(fulfillmentIndex);
    }

    function test__revertMissingFulfillmentComponentOnAggregation() public {
        vm.expectRevert(abi.encodeWithSelector(FulfillmentApplicationErrors.MissingFulfillmentComponent
↪ OnAggregation.selector,
↪ Side.OFFER));
        _revertMissingFulfillmentComponentOnAggregation(Side.OFFER);
    }

    function test__revertOfferAndConsiderationRequiredOnFulfillment() public {

↪ vm.expectRevert(FulfillmentApplicationErrors.OfferAndConsiderationRequiredOnFulfillment.selector);
        _revertOfferAndConsiderationRequiredOnFulfillment();
    }
```

```
}
```

Seaport: Fixed in PR 867.

Spearbit: Verified.

5.4.37 Typo in comment for the selector used in `ConsiderationEncoder.sol#abi_encode_validateOrder()`

Severity: Informational

Context: ConsiderationEncoder.sol#L221, ConsiderationEncoder.sol#L321

Description: Minor typo in comments:

```
        // Write ratifyOrder selector and get pointer to start of calldata
        dst.write(validateOrder_selector);
```

Recommendation:

```
-       // Write ratifyOrder selector and get pointer to start of calldata
+       // Write validateOrder selector and get pointer to start of calldata
        dst.write(validateOrder_selector);
```

Seaport: Fixed in PR 831.

Spearbit: Verified.

5.4.38 `_contractNonces[offerer]` gets incremented even if the `generateOrder(...)`'s return data does not satisfy all the constrainsts.

Severity: Informational

Context:

- OrderValidator.sol#L382
- OrderValidator.sol#L402-L404
- ConsiderationEncoder.sol#L144-L147

Description: `_contractNonces[offerer]` gets incremented even if the `generateOrder(...)`'s return data does not satisfy all the constraints. This is the case when `errorBuffer != 0` and `revertOnInvalid == false` (`fulfillAvailableOrders`, `fulfillAvailableAdvancedOrders`).

In this case, `Seaport` would not call back into the contract `offerer`'s `ratifyOrder(...)` endpoint. Thus, the next time this `offerer` receives a `ratifyOrder(...)` call from `Seaport`, the `nonce` shared with it might have incremented more than `1`.

Recommendation: The above scenario should be documented so that the contract `offerer` would be aware of this fact.

Seaport: Fixed in PR 865.

Spearbit: Verified.

5.4.39 Users need to be cautious about what proxied/modified `Seaport` or `Conduit` instances they approve their tokens to

Severity: Informational

Context:

- ConsiderationBase.sol#L112-L113
- ConsiderationBase.sol#L37

Description: `Seaport` (S) uses `EIP-712` domain separator to make sure that when users sign orders, the signed orders only apply to that specific `Seaport` by pinpointing its `name`, `version`, the `chainid`, and its `address`. The domain separator is calculated and cached once the `Seaport` contract gets deployed. The domain separator only gets recalculated when/if the `chainid` changes (in the case of a hard fork for example). Some actors can take advantage of this caching mechanism by deploying a contract (S') that :

- Delegates some of its endpoints to `Seaport` or it's just a proxy contract.
- Its codebase is almost identical to `Seaport` except that the domain separator actually replicates what the original `Seaport` is using. This only requires 1 or 2 lines of code change (in this case the caching mechanism is not important)

```
function _deriveDomainSeparator() {
   ...
  // Place the address of this contract in the next memory location.
  mstore(FourWords, MAIN_SEAPORT_ADDRESS) // <--- modified line and perhaps the actor can define a
↪  named constant
```

Assume a user approves either:

1. Both the original `Seaport` instance and the modified/proxied instance or,
2. A conduit that has open channels to both the original `Seaport` instance and the modified/proxied instance.

And signs an order for the original `Seaport` that in the 1st case doesn't use any conduits or in the 2nd case the order uses the approved conduit with 2 open channels. Then one can use the same signature once with the original `Seaport` and once with the modified/proxied one to receive more tokens than `offerer` / `user` originally had intended to sell.

Recommendation: The scenario depicted above belongs to the phishing attack category. Users need to be diligent when they approve their tokens to an operator.

OpenSea can monitor the chains to label contracts that resemble its code base but have some modifications or monitor calls that have been delegated to `Seaport`. OpenSea also should monitor all the conduits (and their open channels) that have been created using its corresponding controller.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.4.40 `ZoneInteraction` contains logic for both `zone` and contract `offerer`s

Severity: Informational

Context:

- ZoneInteraction.sol#L34-L39

Description: `ZoneInteraction` contains logic for both `zone` and contract `offerer`s.

Recommendation: Perhaps the contract/file name can be changed to reflect that its functionality is not only restricted to `zone`'s. Also the contract level NatSpec can be updated.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.4.41 Orders of `CONTRACT` order type can lower the value of a token offered

Severity: Informational

Context:

- ContractOffererInterface.sol#L7-L12
- ContractOffererInterface.sol#L16-L22

Description: Sometimes tokens have extra value because of the derived tokens owned by them (for example an accessory for a player in a game). With the introduction of contract `offerer`, one can create a contract `offerer` that automatically lowers the value of a token, for example, by transferring the derived connected token to a different item when `Seaport` calls the `generateOrder(...)`. When such an order is included in a collection of orders the only way to ensure that the recipient of the item will hold a token which value hasn't depreciated during the transaction is that the recipient would also need to use a kind of mirrored order that incorporates either a `CONTRACT` or restricted order type that can do a post-transfer check.

Recommendation: Document scenarios like the above for the users so that they can interact with `Seaport` with the above knowledge in mind.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.4.42 Restricted order checks in case where offerer and the fulfiller are the same

Severity: Informational

Context: ZoneInteraction.sol#L61

Description: Seaport 1.2 disallowed skipping restricted order checks when `offerrer` and fulfiller are the same.

- Remove special-casing for offerer-fulfilled restricted orders: Offerers may currently bypass restricted order checks when fulfilling their own orders. This complicates reasoning about restricted order validation, can aid in the deception of other offerers or fulfillers in some unusual edge cases, and serves little practical use.

However, in the case of the `offerer == fulfiller == zone`, the check continues to be skipped.

Recommendation: Document this edge case.

Seaport: Fixed in PR 854.

Spearbit: Verified.

5.4.43 Clean up inline documentation

Severity: Informational

Context:

- `ConsiderationEncoder.sol#L216`, `ConsiderationEncoder.sol#L316`, `ZoneInteraction.sol#L97-L100`, `ZoneInteraction.sol#L182`
- `ConsiderationStructs.sol#L167`, `ZoneInteraction.sol#L78-L83`, `ZoneInteraction.sol#L45-L49`
- `TransferHelperErrors.sol#L29`, `TransferHelperErrors.sol#L40`
- `Consideration.sol#L39`

Description: The comments highlighted in *Context* need to be removed or updated.

- Remove the following:

```
ConsiderationEncoder.sol:216: // @todo Dedupe some of this
ConsiderationEncoder.sol:316: // @todo Dedupe some of this
ZoneInteraction.sol:97: // bytes memory callData;
ZoneInteraction.sol:100: // function(bytes32) internal view errorHandler;
ZoneInteraction.sol:182: // let magicValue := shr(224, mload(callData))
```

- ConsiderationStructs.sol#L167 and ZoneInteraction.sol#L82 contain an outdated comment about the `extraData` attribute. There is no longer a `staticcall` being done, and the function `isValidOrderIncludingExtraData` no longer exists.
  - The NatSpec comment for `_assertRestrictedAdvancedOrderValidity` mentions:

    ```
    /**
     * @dev Internal view function to determine whether an order is a restricted
     *      order and, if so, to ensure that it was either submitted by the
     *      offerer or the zone for the order, or that the zone returns the
     *      expected magic value upon performing a staticcall to `isValidOrder`
     *      or `isValidOrderIncludingExtraData` depending on whether the order
     *      fulfillment specifies extra data or criteria resolvers.
    ```

    A few of the facts are not correct anymore:

    * This function is not a `view` function anymore and change the storage state either for a `zone` or a contract `offerer`.
    * It is not only for restricted orders but also applies to orders of `CONTRACT` order type.
    * It performs actuall `call`s and not `staticcall`s anymore.
    * it calls the `isValidOrder` endpoint of a `zone` or the `ratifyOrder` endpoint of a contract `offerer` depending on the order type.
    * If it is dealing with a restricted order, the check is only skipped if the `msg.sender` is the `zone`. If `Seaport` is called by the `offerer` for a restricted order, the call to the `zone` is still performed.
  - Same comments apply to `_assertRestrictedBasicOrderValidity` excluding the case when order is of `CONTRACT` order type.
- Typos in `TransferHelperErrors.sol`

```
- * @dev Revert with an error when a call to a ERC721 receiver reverts with
+ * @dev Revert with an error when a call to an ERC721 receiver reverts with
```

- The `@` NatSpec fields have an extra space in `Consideration.sol`:

```
* @ <field>
```

  The extra space can be removed.

Recommendation: Clean up comments.

Seaport: Fixed in PR 816.

Spearbit: Verified.

5.4.44 Consider writing tests for hard coded constants in `ConsiderationConstants.sol`

Severity: Informational

Context: `ConsiderationConstants.sol`, `ConduitConstants.sol`

Description: There are many hard coded constants, most being function selectors, that should be tested against.

Recommendation: Consider adding the following foundry test file: `ConstantsTest.t.sol`

`FOUNDRY_PROFILE=test forge test --match-contract ConstantsTest`

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

import "forge-std/Test.sol";

import "../../contracts/lib/ConsiderationConstants.sol";

import "../../contracts/conduit/lib/ConduitConstants.sol";

import {
    ConduitInterface
} from "../../contracts/interfaces/ConduitInterface.sol";


import {
    ContractOffererInterface
} from "../../contracts/interfaces/ContractOffererInterface.sol";

import {
    EIP1271Interface
} from "../../contracts/interfaces/EIP1271Interface.sol";

import {
    ZoneInterface
} from "../../contracts/interfaces/ZoneInterface.sol";

import {
    FulfillmentApplicationErrors
} from "../../contracts/interfaces/FulfillmentApplicationErrors.sol";

import {
    AmountDerivationErrors
} from "../../contracts/interfaces/AmountDerivationErrors.sol";

import {
    CriteriaResolutionErrors
} from "../../contracts/interfaces/CriteriaResolutionErrors.sol";

import {
    ZoneInteractionErrors
} from "../../contracts/interfaces/ZoneInteractionErrors.sol";

import {
    SignatureVerificationErrors
} from "../../contracts/interfaces/SignatureVerificationErrors.sol";

import {
    TokenTransferrerErrors
} from "../../contracts/interfaces/TokenTransferrerErrors.sol";
```

```
import {
    ReentrancyErrors
} from "../../contracts/interfaces/ReentrancyErrors.sol";

import {
    ConsiderationEventsAndErrors
} from "../../contracts/interfaces/ConsiderationEventsAndErrors.sol";

contract ConstantsTest is Test {

    // ConsiderationConstants.sol

    // Function selectors

    /*
     * function generateOrder(
     *   address fulfiller,
     *   SpentItem[] calldata minimumReceived,
     *   SpentItem[] calldata maximumSpent,
     *   bytes calldata context
     * )
     * Defined in ContractOffererInterface.sol
     */
    function test_generateOrder_selector() public {
        assertEq(generateOrder_selector, uint32(ContractOffererInterface.generateOrder.selector));
    }

    /* function ratifyOrder(
     *   SpentItem[] calldata offer,
     *   ReceivedItem[] calldata consideration,
     *   bytes calldata context,
     *   bytes32[] calldata orderHashes,
     *   uint256 contractNonce
     * )
     * Defined in ContractOffererInterface.sol
     */

    function test_ratifyOrder_selector() public {
        assertEq(ratifyOrder_selector, uint32(ContractOffererInterface.ratifyOrder.selector));
    }

    /* function validateOrder(
     *   ZoneParameters calldata zoneParameters
     * )
     * Defined in ZoneInterface.sol
     */
    function test_validateOrder_selector() public {
        assertEq(validateOrder_selector, uint32(ZoneInterface.validateOrder.selector));
    }

    /* function isValidSignature(
     *   bytes32 digest,
     *   bytes calldata signature
     * )
     * Defined in EIP1271Interface.sol
     */
    function test_isValidSignature_selector() public {
        assertEq(EIP1271_isValidSignature_selector,
↪   bytes32(EIP1271Interface.isValidSignature.selector));
    }

    // Error selectors
```

```
    /* error MissingFulfillmentComponentOnAggregation(uint8 side)
     * Defined in FulfillmentApplicationErrors.sol
     */
    function test_MissingFulfillmentComponentOnAggregation_error_selector() public {
        assertEq(MissingFulfillmentComponentOnAggregation_error_selector,
↪   uint32(FulfillmentApplicationErrors.MissingFulfillmentComponentOnAggregation.selector));
    }

    /* error OfferAndConsiderationRequiredOnFulfillment()
     * Defined in FulfillmentApplicationErrors.sol
     */
    function test_OfferAndConsiderationRequiredOnFulfillment_error_selector() public {
        assertEq(OfferAndConsiderationRequiredOnFulfillment_error_selector,
↪   uint32(FulfillmentApplicationErrors.OfferAndConsiderationRequiredOnFulfillment.selector));
    }

    /* error MismatchedFulfillmentOfferAndConsiderationComponents(uint256 fulfillmentIndex)
     * Defined in FulfillmentApplicationErrors.sol
     */
    function test_MismatchedFulfillmentOfferAndConsiderationComponents_error_selector() public {
        assertEq(MismatchedFulfillmentOfferAndConsiderationComponents_error_selector,
↪   uint32(FulfillmentApplicationErrors.MismatchedFulfillmentOfferAndConsiderationComponents.selector));
    }

    /* error InvalidFulfillmentComponentData()
     * Defined in FulfillmentApplicationErrors.sol
     */
    function test_InvalidFulfillmentComponentData_error_selector() public {
        assertEq(InvalidFulfillmentComponentData_error_selector,
↪   uint32(FulfillmentApplicationErrors.InvalidFulfillmentComponentData.selector));
    }

    /* error InexactFraction()
     * Defined in AmountDerivationErrors.sol
     */
    function test_InexactFraction_error_selector() public {
        assertEq(InexactFraction_error_selector,
↪   uint32(AmountDerivationErrors.InexactFraction.selector));
    }

    /* error OrderCriteriaResolverOutOfRange(uint8 side)
     * Defined in CriteriaResolutionErrors.sol
     */
    function test_OrderCriteriaResolverOutOfRange_error_selector() public {
        assertEq(OrderCriteriaResolverOutOfRange_error_selector,
↪   uint32(CriteriaResolutionErrors.OrderCriteriaResolverOutOfRange.selector));
    }

    /* error UnresolvedOfferCriteria(uint256 orderIndex, uint256 offerIndex)
     * Defined in CriteriaResolutionErrors.sol
     */
    function test_UnresolvedOfferCriteria_error_selector() public {
        assertEq(UnresolvedOfferCriteria_error_selector,
↪   uint32(CriteriaResolutionErrors.UnresolvedOfferCriteria.selector));
    }

    /* error UnresolvedConsiderationCriteria(uint256 orderIndex, uint256 considerationIndex)
     * Defined in CriteriaResolutionErrors.sol
     */
    function test_UnresolvedConsiderationCriteria_error_selector() public {
```

```
        assertEq(UnresolvedConsiderationCriteria_error_selector,
↪   uint32(CriteriaResolutionErrors.UnresolvedConsiderationCriteria.selector));
    }

    /* error OfferCriteriaResolverOutOfRange()
     * Defined in CriteriaResolutionErrors.sol
     */
    function test_OfferCriteriaResolverOutOfRange_error_selector() public {
        assertEq(OfferCriteriaResolverOutOfRange_error_selector,
↪   uint32(CriteriaResolutionErrors.OfferCriteriaResolverOutOfRange.selector));
    }

    /* error ConsiderationCriteriaResolverOutOfRange()
     * Defined in CriteriaResolutionErrors.sol
     */
    function test_ConsiderationCriteriaResolverOutOfRange_error_selector() public {
        assertEq(ConsiderationCriteriaResolverOutOfRange_error_selector,
↪   uint32(CriteriaResolutionErrors.ConsiderationCriteriaResolverOutOfRange.selector));
    }

    /* error CriteriaNotEnabledForItem()
     * Defined in CriteriaResolutionErrors.sol
     */
    function test_CriteriaNotEnabledForItem_error_selector() public {
        assertEq(CriteriaNotEnabledForItem_error_selector,
↪   uint32(CriteriaResolutionErrors.CriteriaNotEnabledForItem.selector));
    }

    /* error InvalidProof()
     * Defined in CriteriaResolutionErrors.sol
     */
    function test_InvalidProof_error_selector() public {
        assertEq(InvalidProof_error_selector, uint32(CriteriaResolutionErrors.InvalidProof.selector));
    }

    /* error InvalidRestrictedOrder(bytes32 orderHash)
     * Defined in ZoneInteractionErrors.sol
     */
    function test_InvalidRestrictedOrder_error_selector() public {
        assertEq(InvalidRestrictedOrder_error_selector,
↪   uint32(ZoneInteractionErrors.InvalidRestrictedOrder.selector));
    }

    /* error InvalidContractOrder(bytes32 orderHash)
     * Defined in ZoneInteractionErrors.sol
     */
    function test_InvalidContractOrder_error_selector() public {
        assertEq(InvalidContractOrder_error_selector,
↪   uint32(ZoneInteractionErrors.InvalidContractOrder.selector));
    }

    /* error BadSignatureV(uint8 v)
     * Defined in SignatureVerificationErrors.sol
     */
    function test_BadSignatureV_error_selector() public {
        assertEq(BadSignatureV_error_selector,
↪   uint32(SignatureVerificationErrors.BadSignatureV.selector));
    }

    /* error InvalidSigner()
     * Defined in SignatureVerificationErrors.sol
     */
```

```
    function test_InvalidSigner_error_selector() public {
        assertEq(InvalidSigner_error_selector,
↪   uint32(SignatureVerificationErrors.InvalidSigner.selector));
    }

    /* error InvalidSignature()
     * Defined in SignatureVerificationErrors.sol
     */
    function test_InvalidSignature_error_selector() public {
        assertEq(InvalidSignature_error_selector,
↪   uint32(SignatureVerificationErrors.InvalidSignature.selector));
    }

    /* error BadContractSignature()
     * Defined in SignatureVerificationErrors.sol
     */
    function test_BadContractSignature_error_selector() public {
        assertEq(BadContractSignature_error_selector,
↪   uint32(SignatureVerificationErrors.BadContractSignature.selector));
    }

    /* error InvalidERC721TransferAmount(uint256 amount)
     * Defined in TokenTransferrerErrors.sol
     */
    function test_InvalidERC721TransferAmount_error_selector() public {
        assertEq(InvalidERC721TransferAmount_error_selector,
↪   uint32(TokenTransferrerErrors.InvalidERC721TransferAmount.selector));
    }

    /* error MissingItemAmount()
     * Defined in TokenTransferrerErrors.sol
     */
    function test_MissingItemAmount_error_selector() public {
        assertEq(MissingItemAmount_error_selector,
↪   uint32(TokenTransferrerErrors.MissingItemAmount.selector));
    }

    /* error UnusedItemParameters()
     * Defined in TokenTransferrerErrors.sol
     */
    function test_UnusedItemParameters_error_selector() public {
        assertEq(UnusedItemParameters_error_selector,
↪   uint32(TokenTransferrerErrors.UnusedItemParameters.selector));
    }

    /* error BadReturnValueFromERC20OnTransfer(address token, address from, address to, uint256 amount)
     * Defined in TokenTransferrerErrors.sol
     */
    function test_BadReturnValueFromERC20OnTransfer_error_selector() public {
        assertEq(BadReturnValueFromERC20OnTransfer_error_selector,
↪   uint32(TokenTransferrerErrors.BadReturnValueFromERC20OnTransfer.selector));
    }

    /* error NoContract(address account)
     * Defined in TokenTransferrerErrors.sol
     */
    function test_NoContract_error_selector() public {
        assertEq(NoContract_error_selector, uint32(TokenTransferrerErrors.NoContract.selector));
    }

    /* error Invalid1155BatchTransferEncoding()
     * Defined in TokenTransferrerErrors.sol
```

```
     */
    function test_Invalid1155BatchTransferEncoding_error_selector() public {
        assertEq(Invalid1155BatchTransferEncoding_error_selector,
↪   uint32(TokenTransferrerErrors.Invalid1155BatchTransferEncoding.selector));
    }

    /* error NoReentrantCalls()
     * Defined in ReentrancyErrors.sol
     */
    function test_NoReentrantCalls_error_selector() public {
        assertEq(NoReentrantCalls_error_selector, uint32(ReentrancyErrors.NoReentrantCalls.selector));
    }

    /* error OrderAlreadyFilled(bytes32 orderHash)
     * Defined in ConsiderationEventsAndErrors.sol
     */
    function test_OrderAlreadyFilled_error_selector() public {
        assertEq(OrderAlreadyFilled_error_selector,
↪   uint32(ConsiderationEventsAndErrors.OrderAlreadyFilled.selector));
    }

    /* error InvalidTime(uint256 startTime, uint256 endTime)
     * Defined in ConsiderationEventsAndErrors.sol
     */
    function test_InvalidTime_error_selector() public {
        assertEq(InvalidTime_error_selector, uint32(ConsiderationEventsAndErrors.InvalidTime.selector));
    }

    /* error InvalidConduit(bytes32 conduitKey, address conduit)
     * Defined in ConsiderationEventsAndErrors.sol
     */
    function test_InvalidConduit_error_selector() public {
        assertEq(InvalidConduit_error_selector,
↪   uint32(ConsiderationEventsAndErrors.InvalidConduit.selector));
    }

    /* error MissingOriginalConsiderationItems()
     * Defined in ConsiderationEventsAndErrors.sol
     */
    function test_MissingOriginalConsiderationItems_error_selector() public {
        assertEq(MissingOriginalConsiderationItems_error_selector,
↪   uint32(ConsiderationEventsAndErrors.MissingOriginalConsiderationItems.selector));
    }

    /* error InvalidCallToConduit(address conduit)
     * Defined in ConsiderationEventsAndErrors.sol
     */
    function test_InvalidCallToConduit_error_selector() public {
        assertEq(InvalidCallToConduit_error_selector,
↪   uint32(ConsiderationEventsAndErrors.InvalidCallToConduit.selector));
    }

    /* error ConsiderationNotMet(uint256 orderIndex, uint256 considerationIndex, uint256
↪   shortfallAmount)
     * Defined in ConsiderationEventsAndErrors.sol
     */
    function test_ConsiderationNotMet_error_selector() public {
        assertEq(ConsiderationNotMet_error_selector,
↪   uint32(ConsiderationEventsAndErrors.ConsiderationNotMet.selector));
    }

    /* error InsufficientEtherSupplied()
```

```
     * Defined in ConsiderationEventsAndErrors.sol
     */
    function test_InsufficientEtherSupplied_error_selector() public {
        assertEq(InsufficientEtherSupplied_error_selector,
↪   uint32(ConsiderationEventsAndErrors.InsufficientEtherSupplied.selector));
    }

    /* error EtherTransferGenericFailure(address account, uint256 amount)
     * Defined in ConsiderationEventsAndErrors.sol
     */
    function test_EtherTransferGenericFailure_error_selector() public {
        assertEq(EtherTransferGenericFailure_error_selector,
↪   uint32(ConsiderationEventsAndErrors.EtherTransferGenericFailure.selector));
    }

    /* error PartialFillsNotEnabledForOrder()
     * Defined in ConsiderationEventsAndErrors.sol
     */
    function test_PartialFillsNotEnabledForOrder_error_selector() public {
        assertEq(PartialFillsNotEnabledForOrder_error_selector,
↪   uint32(ConsiderationEventsAndErrors.PartialFillsNotEnabledForOrder.selector));
    }

    /* error OrderIsCancelled(bytes32 orderHash)
     * Defined in ConsiderationEventsAndErrors.sol
     */
    function test_OrderIsCancelled_error_selector() public {
        assertEq(OrderIsCancelled_error_selector,
↪   uint32(ConsiderationEventsAndErrors.OrderIsCancelled.selector));
    }

    /* error OrderPartiallyFilled(bytes32 orderHash)
     * Defined in ConsiderationEventsAndErrors.sol
     */
    function test_OrderPartiallyFilled_error_selector() public {
        assertEq(OrderPartiallyFilled_error_selector,
↪   uint32(ConsiderationEventsAndErrors.OrderPartiallyFilled.selector));
    }

    /* error InvalidCanceller()
     * Defined in ConsiderationEventsAndErrors.sol
     */
    function test_InvalidCanceller_error_selector() public {
        assertEq(InvalidCanceller_error_selector,
↪   uint32(ConsiderationEventsAndErrors.InvalidCanceller.selector));
    }

    /* error BadFraction()
     * Defined in ConsiderationEventsAndErrors.sol
     */
    function test_BadFraction_error_selector() public {
        assertEq(BadFraction_error_selector, uint32(ConsiderationEventsAndErrors.BadFraction.selector));
    }

    /* error InvalidMsgValue(uint256 value)
     * Defined in ConsiderationEventsAndErrors.sol
     */
    function test_InvalidMsgValue_error_selector() public {
        assertEq(InvalidMsgValue_error_selector,
↪   uint32(ConsiderationEventsAndErrors.InvalidMsgValue.selector));
    }
```

```
    /* error InvalidBasicOrderParameterEncoding()
     * Defined in ConsiderationEventsAndErrors.sol
     */
    function test_InvalidBasicOrderParameterEncoding_error_selector() public {
        assertEq(InvalidBasicOrderParameterEncoding_error_selector,
↪   uint32(ConsiderationEventsAndErrors.InvalidBasicOrderParameterEncoding.selector));
    }

    /* error NoSpecifiedOrdersAvailable()
     * Defined in ConsiderationEventsAndErrors.sol
     */
    function test_NoSpecifiedOrdersAvailable_error_selector() public {
        assertEq(NoSpecifiedOrdersAvailable_error_selector,
↪   uint32(ConsiderationEventsAndErrors.NoSpecifiedOrdersAvailable.selector));
    }

    /* error InvalidNativeOfferItem()
     * Defined in ConsiderationEventsAndErrors.sol
     */
    function test_InvalidNativeOfferItem_error_selector() public {
        assertEq(InvalidNativeOfferItem_error_selector,
↪   uint32(ConsiderationEventsAndErrors.InvalidNativeOfferItem.selector));
    }

    /* error ConsiderationLengthExceedsTotalOriginal()
     * Defined in ConsiderationEventsAndErrors.sol
     */
    function test_ConsiderationLengthExceedsTotalOriginal_error_selector() public {
        assertEq(ConsiderationLengthExceedsTotalOriginal_error_selector,
↪   uint32(ConsiderationEventsAndErrors.ConsiderationLengthExceedsTotalOriginal.selector));
    }

    /* error Panic(uint256 code)
     * Built-in Solidity error
     */
    function test_Panic_error_selector() public {
        assertEq(Panic_error_selector, uint32(bytes4(keccak256("Panic(uint256)"))));
    }

    // uint256 constant MaxUint8 = 0xff;
    function testMaxUint8() public {
        assertEq(MaxUint8, type(uint8).max);
    }

    // uint256 constant MaxUint120 = 0xffffffffffffffffffffffffffffff;
    function testMaxUint120() public {
        assertEq(MaxUint120, type(uint120).max);
    }

    // ConduitConstants.sol

    // Error signature

    /* error ChannelClosed(address channel)
     * Defined in ConduitInterface.sol
     */
    function test_ChannelClosed_error_signature() public {
        assertEq(bytes32(ChannelClosed_error_signature),
↪   bytes32(ConduitInterface.ChannelClosed.selector));
    }
}
```

Seaport: Fixed in PR 885.

Spearbit: Verified.

5.4.45 Unused / Redundant imports in `ZoneInteraction.sol`

Severity: Informational

Context: `ZoneInteraction.sol`

Description: There are multiple unused / redundant imports.

Recommendation: Remove the following imports

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

import { ZoneInterface } from "../interfaces/ZoneInterface.sol";

-import {
-   ContractOffererInterface
-} from "../interfaces/ContractOffererInterface.sol";

-import { ItemType, OrderType } from "./ConsiderationEnums.sol";
+import { OrderType } from "./ConsiderationEnums.sol";

import {
-   AdvancedOrder,
-   OrderParameters,
-   BasicOrderParameters,
-   AdditionalRecipient,
    ZoneParameters,
-  OfferItem,
-   ConsiderationItem,
-   SpentItem,
-  ReceivedItem
} from "./ConsiderationStructs.sol";

import { ZoneInteractionErrors } from "../interfaces/ZoneInteractionErrors.sol";

import { LowLevelHelpers } from "./LowLevelHelpers.sol";

-import "./ConsiderationConstants.sol";

-import "./ConsiderationErrors.sol";
-import "./PointerLibraries.sol";
import "./ConsiderationEncoder.sol";
```

Seaport: Fixed in PR 833.

Spearbit: Verified.

5.4.46 Orders of `CONTRACT` order type do not enforce a usage of a specific `conduit`

Severity: Informational

Context:

- ConsiderationStructs.sol#L149
- ContractOffererInterface.sol#L7-L12
- ContractOffererInterface.sol#L16-L22
- ZoneInteraction.sol#L119-L124
- ConsiderationEncoder.sol#L128
- ConsiderationEncoder.sol#L65
- OrderValidator.sol#L369-L372

Description: None of the endpoints (`generateOrder` and `ratifyOrder`) for an order of `CONTRACT` order type enforce using a specific conduit. A contract `offerer` can enforce the usage of a specific conduit or just `Seaport` by setting allowances or approval for specific tokens. If a `caller` calls into different `Seaport` endpoints and does not provide the correct conduit key, then the order would revert.

Currently, the `ContractOffererInterface` interface does not have a specific endpoint to discover which conduits the contract `offerer` would prefer users to use. `getMetadata()` would be able to return a `metadata` that encodes the conduit key.

For (advanced) orders of not `CONTRACT` order types, the `offerer` would sign the order and the conduit key is included in the signed hash. Thus, the conduit is enforced whenever that order gets included in a collection by an actor calling `Seaport`.

Recommendation: The above points would need to be documented/commented for the users and perhaps in `ContractOffererInterface` we can add an endpoint for discovering conduit keys.

Seaport: Fixed in PR 887.

Spearbit: Verified.

5.4.47 Calls to `Seaport` that would fulfill or match a collection of advanced orders can be front-ran to claim any unused offer items

Severity: Informational

Context:

- Consideration.sol#L222
- Consideration.sol#L320
- Consideration.sol#L382
- Consideration.sol#L435

Description: Calls to `Seaport` that would fulfill or match a collection of advanced orders can be front-ran to claim any unused offer items. These endpoints include:

- `fulfillAvailableOrders`
- `fulfillAvailableAdvancedOrders`
- `matchOrders`
- `matchAdvancedOrders`

Anyone can monitor the mempool to find calls to the above endpoints and calculate if there are any unused offer item amounts. If there are unused offer item amounts, the actor can create orders with no offer items, but with consideration items mirroring the unused offer items and populate the fulfillment aggregation data to match the

unused offer items with the new mirrored consideration items. It is possible that the call by the actor would be successful under certain conditions. For example, if there are orders of `CONTRACT` order type involved, the contract `offerer` might reject this actor (the rejection might also happen by the `zone`s used when validating the order). But in general, this strategy can be implemented by anyone.

Recommendation: The above scenario should be documented and perhaps monitored.

Seaport: Fixed in PR 886.

Spearbit: Verified.

5.4.48 Advance orders of `CONTRACT` order types can generate orders with more offer items and the extra offer items might not end up being used.

Severity: Informational

Context:

- OrderValidator.sol#L410-L413

Description: When `Seaport` gets a collection of advanced orders to fulfill or match, if one of the orders has a `CONTRACT` order type, `Seaport` calls the `generateOrder(...)` endpoint of that order's `offerer`. `generateOrder(...)` can provide extra offer items for this order. These extra offer items might have not been known beforehand by the `caller`. And if the `caller` would not incorporate the indexes for the extra items in the fulfillment aggregation data, the extra items would end up not being aggregated into any executions.

Recommendation: The extra offer items provided by a contract `offerer` might not end up being used when fulfilling or matching a collection of advanced orders. The `caller` would need knowledge of these extra items beforehand to incorporate them. If the contract `offerer` implements `previewOrder` which would return the same data as `generateOrder` when a shared input data is provided to them, then the `caller` can predict the fulfillment aggregation data.

This scenario should be documented regardless.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.4.49 Typo for the index check comment in `_aggregateValidFulfillmentConsiderationItems`

Severity: Informational

Context:

- FulfillmentApplier.sol#L566

Description: There is a typo in `_aggregateValidFulfillmentConsiderationItems`:

```
// Retrieve item index using an offset of the fulfillment pointer.
let itemIndex := mload(
    add(mload(fulfillmentHeadPtr), Fulfillment_itemIndex_offset)
)

// Ensure that the order index is not out of range.  <---------- the line with typo
if iszero(lt(itemIndex, mload(considerationArrPtr))) {
    throwInvalidFulfillmentComponentData()
}
```

The `itemIndex` above refers to the index in consideration array and not the order.

Recommendation:

```
- // Ensure that the order index is not out of range.
+ // Ensure that the consideration item index is not out of range.
```

Seaport: Fixed in PR 819.

Spearbit: Verified.

5.4.50 Document the unused parameters for orders of `CONTRACT` order type

Severity: Informational

Context:

- ConsiderationDecoder.sol#L569-L574
- OrderFulfiller.sol#L381

Description: If an advance order `advancedOrder` is of `CONTRACT` order type, certain parameters are not being used in the code base, specifically:

- `numerator`: only used for skipping certain operations (see *`AdvancedOrder.numerator` and `AdvancedOrder.denominator` are unchecked for orders of `CONTRACT` order type*)
- `denominator`: --
- `signature`: --
- `parameters.zone`: only used when emitting the `OrderFulfilled` event.
- `parameters.offer.endAmount`: `endAmount` and `startAmount` for offer items will be set to the `amount` sent back by `generateOrder` for the corresponding item.
- `parameters.consideration.endAmount`: `endAmount` and `startAmount` for consideration items will be set to the `amount` sent back by `generateOrder` for the corresponding item
- `parameters.consideration.recipient`: the `offerer` contract returns new recipients when `generateOrder` gets called
- `parameters.zoneHash`: --
- `parameters.salt`: --
- `parameters.totalOriginalConsiderationItems`: --

Recommendation: Document the decision for not including these parameters.

Seaport: Acknowledged.

Spearbit: Acknowledged.

5.4.51 The check against `totalOriginalConsiderationItems` is skipped for orders of `CONTRACT` order type

Severity: Informational

Context: Assertions.sol#L54-L57

Description: The following inequality is skipped for orders of `CONTRACT` order type which compares `AdvancedOrder.parameters.totalOriginalConsiderationItems` with `AdvancedOrder.parameters.consideration.length`:

```
// Ensure supplied consideration array length is not less than the original.
if (suppliedConsiderationItemTotal < originalConsiderationItemTotal) {
    _revertMissingOriginalConsiderationItems();
}
```

Recommendation: Leave a comment/document that the above inequality is skipped for `CONTRACT` order types.

Seaport: Fixed in commit af89836.

Spearbit: Verified.

5.4.52 `getOrderStatus` returns the default values for `orderHash` that is derived for orders of `CONTRACT` order type

Severity: Informational

Context:

- Consideration.sol#L548

Description: Since the `_orderStatus[orderHash]` does not get set for orders of `CONTRACT` order type, `getOrderStatus` would always returns `(false, false, 0, 0)` for those hashes (unless there is a hash collision with other types of orders)

Recommendation: This can scenario can be documented in the NatSpec comments for `getOrderStatus` or make sure to update `_orderStatus[orderHash]` for orders of `CONTRACT` order type.

Seaport: Fixed in PR 835.

Spearbit: Verified.

5.4.53 `validate` skips `CONTRACT` order types but `cancel` does not

Severity: Informational

Context:

- Consideration.sol#L488
- Consideration.sol#L466
- OrderValidator.sol#L592-L595

Description: When validating orders `validate` skips any order of `CONTRACT` order type, but `cancel` does not skip these order types.

When fulfilling or matching orders for `CONTRACT` order types, `_orderStatus` does not get checked or populated. But in `cancel` the `isValidated` and the `isCancelled` fields get set. This is basically a no-op for these order types.

Recommendation: To be consistent with the validation endpoint, `cancel` can also skip updating the `_orderStatus` for orders of the `CONTRACT` order type. The skipping check might add some gas overhead for when the order is not of this type, but when it is, this will save us some gas.

Seaport: Fixed in PR 853.

Spearbit: Verified.

5.4.54 The literal `0x1c` used as the starting offset of a custom error in a `revert` statement can be replaced by the named constant `Error_selector_offset`

Severity: Informational

Context:

- AmountDeriver.sol#L128
- Assertions.sol#L99
- Executor.sol#L255
- FulfillmentApplier.sol#L233
- FulfillmentApplier.sol#L243
- FulfillmentApplier.sol#L486
- FulfillmentApplier.sol#L522
- FulfillmentApplier.sol#L532

- FulfillmentApplier.sol#L761
- OrderValidator.sol#L270
- SignatureVerification.sol#L219
- SignatureVerification.sol#L228
- SignatureVerification.sol#L256
- SignatureVerification.sol#L263
- SignatureVerification.sol#L291
- TokenTransferrer.sol#L221
- TokenTransferrer.sol#L263
- TokenTransferrer.sol#L353
- TokenTransferrer.sol#L392
- TokenTransferrer.sol#L495
- TokenTransferrer.sol#L571
- ConsiderationConstants.sol#L410

Description: In the context above, `0x1c` is used to signal the start of a custom error block saved in the memory:

```
revert(0x1c, _LENGTH_)
```

For the above literal, we also have a named constant defined in `ConsiderationConstants.sol#L410`:

```
uint256 constant Error_selector_offset = 0x1c;
```

The named constant `Error_selector_offset` has been used in most places that a custom error is reverted in an assembly block.

Recommendation: The literal `0x1c` used as the starting offset of a custom error `revert` can be replaced by the named constant `Error_selector_offset`:

```
import { Error_selector_offset } from "./ConsiderationConstants.sol";
...
   revert(Error_selector_offset, _LENGTH_)
```

Seaport: Fixed in PR 813.

Spearbit: Verified.