Introducing Code4rena Blue: Dedicated defense. Competitive bounties. Independent judging.
Learn more →

OpenSea Seaport 1.2 contest Findings & Analysis Report

2023-03-21

Table of contents

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit contest outlined in this document, C4 conducted an analysis of the OpenSea Seaport 1.2 smart contract system written in Solidity. The audit contest took place between January 13—January 23 2023.

Wardens

23 Wardens contributed reports to the OpenSea Seaport 1.2 contest:

1. 0xsomeone
2. 0xSmartContract

3. ABA
4. Chom
5. Dravee
6. IIIIIII
7. Josiah
8. RaymondFam
9. Rickard
10. Rolezn
11. atharvasama
12. brgltd
13. btk
14. c3phas
15. chaduke
16. charlesjhongc
17. csanuragjain
18. delfin454000
19. horsefacts
20. karanctf
21. nadin
22. oyc_109
23. saneryee

This contest was judged by hickuphh3.

Final report assembled by liveactionllama.

Summary

The C4 analysis yielded an aggregated total of 1 unique vulnerabilities. Of these vulnerabilities, 0 received a risk rating in the category of HIGH severity and 1 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 17 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 9 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the [C4 OpenSea Seaport 1.2 contest repository](), and is composed of 54 smart contracts written in the Solidity programming language and includes 10,087 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities based on three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

For more information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on [the C4 website](), specifically our section on [Severity Categorization]().

Medium Risk Findings (1)

[M-01] Incorrect Encoding of Order Hashes

Submitted by [0xsomeone]()

[contracts/lib/ConsiderationEncoder.sol#L569-L574]()

The order hashes are incorrectly encoded during the `_encodeOrderHashes` mechanism, causing functions such as `_encodeRatifyOrder` and `_encodeValidateOrder` to misbehave.

Proof of Concept

The order hashes encoding mechanism appears to be incorrect as the instructions `srcLength.next().offset(headAndTailSize)` will cause the pointer to move to the end of the array (i.e. `next()` skips the array's `length` bitwise entry and `offset(headAndTailSize)` causes the pointer to point right after the last element). In turn, this will cause the `0x04` precompile within `MemoryPointerLib::copy` to handle the data incorrectly and attempt to copy data from the `srcLength.next().offset(headAndTailSize)` pointer onwards which will be un-allocated space and thus lead to incorrect bytes being copied.

Tools Used

Manual inspection of the codebase, documentation of the ETH precompiles, and the Solidity compiler documentation.

Recommended Mitigation Steps

We advise the `offset` instruction to be omitted as the current implementation will copy from unsafe memory space, causing data corruption in the worst-case scenario and incorrect order hashes being specified in the encoded payload. As an additional point, the `_encodeOrderHashes` will fail execution if the array of order hashes is empty as a `headAndTailSize` of `0` will cause the `MemoryPointerLib::copy` function to fail as the precompile would yield a `returndatasize()` of `0`.

0age (OpenSea) confirmed, but disagreed with severity and commented:

> This is a confirmed issue (though categorizing it as high-risk seems unfair. At worst, it just means that zones and contract offerers wouldn't be able to rely on the orderHashes array) and has been fixed here: https://github.com/ProjectOpenSea/seaport/pull/918

hickuphh3 (judge) decreased severity to Medium and commented:

> Agree that high severity is overstated. Given that it would affect upstream functions (`_encodeRatifyOrder` and `_encodeValidateOrder` is called by a few other functions like `_assertRestrictedAdvancedOrderValidity()`), medium severity would be more appropriate.

> 2 — Med: Assets not at direct risk, but the function of the protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions, but external requirements.

Low Risk and Non-Critical Issues

For this contest, 17 reports were submitted by wardens detailing low risk and non-critical issues. The **report highlighted below** by **horsefacts** received the top score from the judge.

The following wardens also submitted reports: **delfin454000**, **Josiah**, **Chom**, **charlesjhongc**, **nadin**, **IIIIIII**, **0xSmartContract**, **csanuragjain**, **brgltd**, **chaduke**, **RaymondFam**, **Rolezn**, **Rickard**, **ABA**, **btk**, *and* **oyc_109**.

[N-01] Replace “ETH” with “Native token”

Seaport 1.2. has mostly replaced references to “ETH” in comments and function names with “native token,” but there are a few exceptions. Consider replacing the following usages of “ETH” with “native token” or similar.

`Seaport.sol`:

```
    * @notice Seaport is a generalized ETH/ERC20/ERC721/ERC1155 mar
    *         lightweight methods for common routes as well as more
    *         methods for composing advanced orders or groups of or
    *         contains an arbitrary number of items that may be spe
    *         along with an arbitrary number of items that must be
    *         the indicated recipients (the "consideration").
    */
```

`SeaportInterface.sol`:

```
    * @notice Seaport is a generalized ETH/ERC20/ERC721/ERC1155 mar
    *         minimizes external calls to the greatest extent possi
    *         lightweight methods for common routes as well as more
    *         methods for composing advanced orders.
    *
```

Consideration.sol:

```
    * @notice Consideration is a generalized ETH/ERC20/ERC721/ERC11
    *         that provides lightweight methods for common routes a
    *         flexible methods for composing advanced orders or grc
    *         Each order contains an arbitrary number of items that
    *         (the "offer") along with an arbitrary number of items
    *         received back by the indicated recipients (the "consi
    */
```

ConsiderationInterface.sol:

```
    * @notice Consideration is a generalized ETH/ERC20/ERC721/ERC11
    *         It minimizes external calls to the greatest extent po
    *         provides lightweight methods for common routes as wel
    *         flexible methods for composing advanced orders.
    *
```

ConsiderationEventsAndErrors.sol:

```
        /**
         * @dev Revert with an error when attempting to fulfill an o
         *      offer for ETH outside of matching orders.
         */
        error InvalidNativeOfferItem();
    }
```

BasicOrderFulfiller#_validateAndFulfillBasicOrder:

```
                // If route > 1 additionalRecipient items are ERC20
```

```
            additionalRecipientsItemType := gt(route, 1)
```

BasicOrderFulfiller#_validateAndFulfillBasicOrder:

```
                    // If route > 2, receivedItemType is route - 2.
                    // the receivedItemType is ERC20 (1). Otherwise,
                    receivedItemType := byte(route, BasicOrder_recei
```

Executor#_transferNativeTokens:

```
            assembly {
                // Transfer the ETH and store if it succeeded or not
                success := call(gas(), to, amount, 0, 0, 0, 0)
            }
```

ConsiderationEventsAndErrors#InsufficientEtherSupplied:

```
        /**
         * @dev Revert with an error when insufficient ether is supp
         *      msg.value when fulfilling orders.
         */
        error InsufficientEtherSupplied();
```

ConsiderationConstants.sol:

```
    /*
     *  error InsufficientEtherSupplied()
     *    - Defined in ConsiderationEventsAndErrors.sol
     *  Memory layout:
     *    - 0x00: Left-padded selector (data begins at 0x1c)
     * Revert buffer is memory[0x1c:0x20]
     */
    uint256 constant InsufficientEtherSupplied_error_selector = 0x1a
    uint256 constant InsufficientEtherSupplied_error_length = 0x04;
```

ConsiderationErrors.sol:

```
    /**
     * @dev Reverts the current transaction with an "InsufficientEth
     *      error message.
     */
    function _revertInsufficientEtherSupplied() pure {
        assembly {
            // Store left-padded selector with push4 (reduces bytecc
            // mem[28:32] = selector
            mstore(0, InsufficientEtherSupplied_error_selector)

            // revert(abi.encodeWithSignature("InsufficientEtherSupp
            revert(Error_selector_offset, InsufficientEtherSupplied_
        }
    }
```

ConsiderationEventsAndErrors.sol:

```
        /**
         * @dev Revert with an error when an ether transfer reverts.
         */
        error EtherTransferGenericFailure(address account, uint256 a
```

ConsiderationConstants.sol:

```
    /*
     *  error EtherTransferGenericFailure(address account, uint256 a
     *    - Defined in ConsiderationEventsAndErrors.sol
     *  Memory layout:
     *    - 0x00: Left-padded selector (data begins at 0x1c)
     *    - 0x20: account
     *    - 0x40: amount
     * Revert buffer is memory[0x1c:0x60]
     */
    uint256 constant EtherTransferGenericFailure_error_selector = 0x
    uint256 constant EtherTransferGenericFailure_error_account_ptr =
    uint256 constant EtherTransferGenericFailure_error_amount_ptr =
    uint256 constant EtherTransferGenericFailure_error_length = 0x44
```

Executor.sol:

```
            // Otherwise, revert with a generic error message.
            assembly {
                // Store left-padded selector with push4, mem[28
                mstore(0, EtherTransferGenericFailure_error_sele
                mstore(EtherTransferGenericFailure_error_account
                mstore(EtherTransferGenericFailure_error_amount_

                // revert(abi.encodeWithSignature(
                //   "EtherTransferGenericFailure(address,uint25
                // )
                revert(
                    Error_selector_offset,
                    EtherTransferGenericFailure_error_length
                )
            }
```

[N-02] Extract or use named constants

The Seaport codebase has done an impressive job avoiding “magic numbers” and using named constants, which makes inline assembly much easier to read, understand, and verify. However, there are a few remaining numbers that could be replaced with more readable named constants.

The `malloc` free function in `PointerLibraries.sol` can use `FreeMemoryPointerSlot` in place of `0x40`:

PointerLibraries#malloc:

```
    /// @dev Allocates `size` bytes in memory by increasing the free
    ///    and returns the memory pointer to the first byte of the a
    // (Free functions cannot have visibility.)
    // solhint-disable-next-line func-visibility
    function malloc(uint256 size) pure returns (MemoryPointer mPtr)
        assembly {
            mPtr := mload(0x40)
            mstore(0x40, add(mPtr, size))
        }
    }
```

Calldata readers in `PointerLibraries.sol` can use `OneWord` in place of `0x20`:

CallDataReaders#readBool

```
        /// @dev Reads the bool at `rdPtr` in returndata.
        function readBool(
            ReturndataPointer rdPtr
        ) internal pure returns (bool value) {
            assembly {
                returndatacopy(0, rdPtr, 0x20)
                value := mload(0)
            }
        }
```

(Note that `returndatacopy(0, rdPtr, 0x20)` is repeated in every `CallDataReaders#readType` function.)

`CalldataPointerLib#next` can use `OneWord` in place of `32`:

CalldataPointerLib#next

```
        /// @dev Returns the calldata pointer one word after `cdPtr`
        function next(
            CalldataPointer cdPtr
        ) internal pure returns (CalldataPointer cdPtrNext) {
            assembly {
                cdPtrNext := add(cdPtr, 32)
            }
        }
```

Similar usages:

- MemoryPointerLib#next
- ReturnDataPointerLib#next

`OrderCombiner` iterates in increments of 32, which could be replaced with `OneWord`:

OrderCombiner:

```
            // Determine the memory offset to terminate on durir
            terminalMemoryOffset = (totalOrders + 1) * 32;
```

OrderCombiner:

```
            // Iterate over each order.
            for (uint256 i = 32; i < terminalMemoryOffset; i +=
                // Retrieve order using assembly to bypass out-c
                assembly {
                    advancedOrder := mload(add(advancedOrders, i
                }
```

[N-03] Fragile check for contract order type

The `OrderType` enum defines five order types. Only one of these represents contract orders:

ConsiderationEnums.sol:

```
    enum OrderType {
        // 0: no partial fills, anyone can execute
        FULL_OPEN,

        // 1: partial fills supported, anyone can execute
        PARTIAL_OPEN,

        // 2: no partial fills, only offerer or zone can execute
        FULL_RESTRICTED,

        // 3: partial fills supported, only offerer or zone can exec
        PARTIAL_RESTRICTED,

        // 4: contract order type
        CONTRACT
    }
```

OrderCombiner#_validateOrdersAndPrepareToFulfill defines non-contract orders as any order with a type less than 4:

```
        {
            // Create a variable indicating if the order
            // contract order. Cache in scratch space to
            // depth errors.
            OrderType orderType = advancedOrder.paramete
            assembly {
                let isNonContract := lt(orderType, 4)
                mstore(0, isNonContract)
            }
        }
```

This is fine for now, but could be fragile: if an additional type is added in the future, it may break this implicit assumption. Consider checking for an exact match against order type 4, which is more robust:

```
        {
            // Create a variable indicating if the order
            // contract order. Cache in scratch space to
            // depth errors.
            OrderType orderType = advancedOrder.paramete
            assembly {
                let isNonContract := iszero(eq(orderType
                mstore(0, isNonContract)
            }
        }
```

`OrderFulfiller#_applyFractionsAndTransferEach` performs a similar check using `lt(orderType, 4)`:

```
        // If non-contract order has native offer items, thr
        {
            OrderType orderType = orderParameters.orderType;
            uint256 invalidNativeOfferItem;
            assembly {
                invalidNativeOfferItem := and(
                    lt(orderType, 4),
                    anyNativeItems
                )
            }
            if (invalidNativeOfferItem != 0) {
```

```
                    _revertInvalidNativeOfferItem();
                }
            }
```

[N-04] Inconsistent use of hex vs. decimal values

Almost all values except for bit shifts are defined in hex, with the following few exceptions:

`CalldataPointerLib` uses `32` rather than `0x20` in a few places:

`CalldataPointerLib#next`

```
    /// @dev Returns the calldata pointer one word after `cdPtr`.
    function next(
        CalldataPointer cdPtr
    ) internal pure returns (CalldataPointer cdPtrNext) {
        assembly {
            cdPtrNext := add(cdPtr, 32)
        }
    }
```

Similar usages:

- `MemoryPointerLib#next`
- `ReturnDataPointerLib#next`

Two lengths in `ConsiderationConstants`:

`NameLengthPtr`:

```
    uint256 constant NameLengthPtr = 77;
```

`Selector_length`:

```
    uint256 constant Selector_length = 4;
```

Precompile addresses:

PointerLibraries#IdentityPrecompileAddress:

```
    uint256 constant IdentityPrecompileAddress = 4;
```

ConsiderationConstants.sol:

```
    address constant IdentityPrecompile = address(4);
```

ConsiderationConstants.sol:

```
    uint256 constant Ecrecover_precompile = 1;
```

Consider converting all of these to hex to enhance readability.

[N-05] Custom comment typos

There are two `@custom:name` comments on functions in `Consideration.sol` that are meant to annotate unnamed input arguments, but are incorrectly annotating the function's return type:

Consideration#validate:

```
        function validate(
            Order[] calldata
        )
            external
            override
            returns (
                /**
                 * @custom:name orders
```

```
             */
            bool /* validated */
        )
    {
        return
            _validate(_toOrdersReturnType(_decodeOrders)(Calldat
    }
```

Consideration#getOrderHash:

```
    function getOrderHash(
        OrderComponents calldata
    )
        external
        view
        override
        returns (
            /**
             * @custom:name order
             */
            bytes32 orderHash
        )
    {
        CalldataPointer orderPointer = CalldataStart.pptr();

        // Derive order hash by supplying order parameters alonc
        orderHash = _deriveOrderHash(
            _toOrderParametersReturnType(
                _decodeOrderComponentsAsOrderParameters
            )(orderPointer),
            // Read order counter
            orderPointer.offset(OrderParameters_counter_offset).
        );
    }
```

[N-06] `AlmostOneWord` is confusing

I find the `AlmostOneWord` constant, which is equal to 31 bytes, pretty confusing in context, since it's not clear from the name what it means to be equal to "almost one word." Consider whether `ThirtyOneBytes` or similar might be a clearer name.

[N-07] Typos in comments

The default order numerator + denominator values are *always* 1 and 1, so this `e.g.` in `ConsiderationDecoder.sol` should be an `i.e.` :

```
// Write default Order numerator and denominator values
mPtr.offset(AdvancedOrder_numerator_offset).write(1);
mPtr.offset(AdvancedOrder_denominator_offset).write(1);
```

[N-08] Duplicated constants

`TypeHashDirectory` defines several constants, like `OneWord`, `OneWordShift`, `AlmostOneWord`, and `FreeMemoryPointerSlot` that are defined elsewhere in the codebase. Consider extracting these to a shared constants file:

```
uint256 internal constant OneWord = 0x20;
uint256 internal constant OneWordShift = 5;
uint256 internal constant AlmostOneWord = 0x1f;
uint256 internal constant FreeMemoryPointerSlot = 0x40;
```

0age (OpenSea) commented:

> This is a high-quality QA report 👍

hickuphh3 (judge) commented:

> 8 non-criticals, but I think they provide more value than the other QA reports I've come across thus far. Hence, it's worthy of an A grade (+bonus from sponsor for flagging it as high-quality).

0age (OpenSea) resolved:

> **[N-01] Replace "ETH" with "Native token":**
> https://github.com/ProjectOpenSea/seaport/pull/921

> **[N-02] Extract or use named constants:**
> https://github.com/ProjectOpenSea/seaport/pull/922

> [N-03] Fragile check for contract order type:
> https://github.com/ProjectOpenSea/seaport/pull/922

> [N-04] Inconsistent use of hex vs. decimal values:
> https://github.com/ProjectOpenSea/seaport/pull/922

> [N-05] Custom comment typos:
> https://github.com/ProjectOpenSea/seaport/pull/924

> [N-06] AlmostOneWord is confusing:
> https://github.com/ProjectOpenSea/seaport/pull/923

> [N-07] Typos in comments:
> https://github.com/ProjectOpenSea/seaport/pull/924

> [N-08] Duplicated constants:
> https://github.com/ProjectOpenSea/seaport/pull/922

Gas Optimizations

For this contest, 9 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **Dravee** received the top score from the judge.

The following wardens also submitted reports: atharvasama, c3phas, 0xSmartContract, IIIIIII, karanctf, RaymondFam, Rolezn, *and* saneryee.

Overview

Risk Rating	Number of issues	Estimated savings	
Gas Issues	6	Around 650	

Codebase Impressions

The codebase is amazingly optimized, as expected from OpenSea. All storage operations are well done and even take into account the SLOT packings (like here where it isn't obvious and the regular dev would've moved the SSTORE into the condition or here where `memory` would've cost more).

The suggestions down below took some research: everything has been justified with POCs, code and even the opcodes on the stack when necessary.

Table of Contents

[G-01] Using XOR (`^`) and OR (`|`) bitwise equivalents

Estimated savings: 73 gas
Max savings according to `yarn profile` *: 282 gas*

On Remix, given only `uint256` types, the following are logical equivalents, but don't cost the same amount of gas:

- `(a != b || c != d || e != f)` costs 571
- `((a ^ b) | (c ^ d) | (e ^ f)) != 0` costs 498 (saving 73 gas)

Consider rewriting as following to save gas:

```
File: FulfillmentApplier.sol
93:         if (
- 94:             execution.item.itemType != considerationItem.i
- 95:             execution.item.token != considerationItem.toke
- 96:             execution.item.identifier != considerationItem
+ 94:             ((uint8(execution.item.itemType) ^ uint8(consi
+ 95:             (uint160(execution.item.token) ^ uint160(consi
+ 96:             (execution.item.identifier ^ considerationItem
97:         ) {
```

Logic POC

Given 4 variables `a`, `b`, `c` and `d` represented as such:

```
0 0 0 0 0 1 1 0 <- a
0 1 1 0 0 1 1 0 <- b
0 0 0 0 0 0 0 0 <- c
1 1 1 1 1 1 1 1 <- d
```

To have `a == b` means that every `0` and `1` match on both variables. Meaning that a XOR (operator `^`) would evaluate to 0 (`(a ^ b) == 0`), as it excludes by definition any equalities.
Now, if `a != b`, this means that there's at least somewhere a `1` and a `0` not matching between `a` and `b`, making `(a ^ b) != 0`.

Both formulas are logically equivalent and using the XOR bitwise operator costs actually the same amount of gas:

```
function xOrEquivalence(uint a, uint b) external returns (
  //return a != b; //370
  //return a ^ b != 0; //370
```

However, it is much cheaper to use the bitwise OR operator (`|`) than comparing the truthy or falsy values:

```
function xOrOrEquivalence(uint a, uint b, uint c, uint d) ex
    //return (a != b || c != d); // 495
    //return (a ^ b | c ^ d) != 0; // 442
}
```

These are logically equivalent too, as the OR bitwise operator (`|`) would result in a `1` somewhere if any value is not `0` between the XOR (`^`) statements, meaning if any XOR (`^`) statement verifies that its arguments are different.

Coded Proof of Concept

This little POC (use `forge test -m test_XorEq`) also proves that the formulas are equivalent:

```
    function test_XorEq(uint8 a, uint8 b, address c, address d,
        assert((a != b || c != d || e != f) == (((a ^ b) | (uint
    }
```

Please keep in mind that Foundry cannot currently fuzz `Enum` types, which is why we're using `uint8` types above, which is [treated the same according to the Solidity documentation](). However, you can try the following test on Remix to make sure, as it will always pass the asserts:

```
    function test_enum(ItemType a, ItemType b) public {
        assert((a != b) == (uint8(a) != uint8(b)));
        assert((a != b) == ((uint8(a) ^ uint8(b)) != 0));
    }
```

yarn profile

This is the diff between the contest repo's `yarn profile` and the added suggestion's `yarn profile`, as `yarn profile` never changes the "Previous Report" it compares the "Current Report" to:

```
=================================================================
| method                          |          min |           max
=================================================================
- | matchAdvancedOrders            | +12 (+0.01%) |     -12 (0%)
+ | matchAdvancedOrders            | -40 (-0.02%) |  -92 (-0.03%
- | matchOrders                    | -12 (-0.01%) | -24 (-0.01%)
+ | matchOrders                    | -20 (-0.01%) | -176 (-0.05%
- | validate                       |        53206 |        83915
+ | validate                       |        53206 |  -24 (-0.03%
=================================================================
- | runtime size                   |        23583 |
+ | runtime size                   | -13 (-0.06%) |
- | init code size                 | +78 (+0.29%) |
+ | init code size                 | +65 (+0.24%) |
=================================================================
```

Added together, the max gas saving counted here is 282.

Consider applying the suggested equivalence and **add a comment mentioning what this is equivalent to**, as this is less human-readable, but still understandable once it's been taught.

[G-02] Shift left by 5 instead of multiplying by 32

Estimated savings: 22 gas
Max savings according to `yarn profile` *: 98 gas*

The equivalent of multiplying by 32 is shifting left by 5. On Remix, a simple POC shows some by replacing one with the other (Optimizer at 10k runs):

```
    function shiftLeft5(uint256 a) public pure returns (uint256)
        //unchecked { return a * 32; } //346
        //unchecked { return a << 5; } //344
    }
```

This is due to the fact that the MUL opcode costs 5 gas and the SHL opcode costs 3 gas. Therefore, saving those 2 units of gas is expected.

Places where this optimization can be applied are as such:

- A simple multiplication by 32:

```
File: OrderCombiner.sol
- 220:             terminalMemoryOffset = (totalOrders + 1) * 32
+ 220:             terminalMemoryOffset = (totalOrders + 1) << 5
```

- Multiplying by the constant `OneWord == 0x20` , as `0x20` in hex is actually `32` in decimals:

```
seaport/contracts/lib/ConsiderationDecoder.sol:
-  386:             uint256 tailOffset = arrLength * OneWord;
+  386:             uint256 tailOffset = arrLength << 5;
-  427:             uint256 arrSize = (arrLength + 1) * OneWord;
+  427:             uint256 arrSize = (arrLength + 1) << 5;
-  485:             uint256 tailOffset = arrLength * OneWord;
```

```
+  485:              uint256 tailOffset = arrLength << 5;
-  525:              uint256 tailOffset = arrLength * OneWord;
+  525:              uint256 tailOffset = arrLength << 5;
-  617:              uint256 tailOffset = arrLength * OneWord;
+  617:              uint256 tailOffset = arrLength << 5;
-  660:              uint256 tailOffset = arrLength * OneWord;
+  660:              uint256 tailOffset = arrLength << 5;
-  731:              uint256 tailOffset = arrLength * OneWord;
+  731:              uint256 tailOffset = arrLength << 5;

seaport/contracts/lib/ConsiderationEncoder.sol:
-  567:              uint256 headAndTailSize = length * OneWord;
+  567:              uint256 headAndTailSize = length << 5;
-  678:              MemoryPointer srcHeadEnd = srcHead.offset(le
+  678:              MemoryPointer srcHeadEnd = srcHead.offset(le
```

Proof of Concept

- Run `forge test -m test_shl5`:

```
    function test_shl5(uint256 a) public {
        vm.assume(a <= type(uint256).max / 32); // This is to av
        assert((a * 32) == (a << 5)); // always true
    }
```

Consider also adding a constant so that the code can be maintainable (`OneWordShiftLength` ?)

yarn profile

```
=================================================================
| method                          |          min |           max
=================================================================
- | matchAdvancedOrders            | +12 (+0.01%) |     -12 (0%)
+ | matchAdvancedOrders            | -84 (-0.05%) |     -12 (0%)
- | matchOrders                    | -12 (-0.01%) | -24 (-0.01%)
+ | matchOrders                    | -12 (-0.01%) | -24 (-0.01%)
```

Added together, the max gas saving counted here is 98.

[G-03] Using a positive conditional flow to save a NOT opcode

Estimated savings: 3 gas

Max savings according to `yarn profile` *: 150 gas*

The following function either revert or returns some value. To save some gas (NOT opcode costing 3 gas), switch to a positive statement:

```
File: OrderValidator.sol
863:     function _revertOrReturnEmpty(
864:         bool revertOnInvalid,
865:         bytes32 contractOrderHash
866:     )
867:         internal
868:         pure
869:         returns (bytes32 orderHash, uint256 numerator, uint
870:     {
- 871:         if (!revertOnInvalid) { //@audit-issue save the N
+ 871:         if (revertOnInvalid) {
- 872:             return (contractOrderHash, 0, 0);
+ 872:             _revertInvalidContractOrder(contractOrderHash
873:         }
874:
- 875:         _revertInvalidContractOrder(contractOrderHash);
+ 875:         return (contractOrderHash, 0, 0);
876:     }
```

yarn profile

```
================================================================
| method                          |          min |          max |
================================================================
- | cancel                         |        41219 |        58403
+ | cancel                         | -12 (-0.03%) |        58403
- | fulfillAdvancedOrder           | +12 (+0.01%) |       225187
+ | fulfillAdvancedOrder           | +12 (+0.01%) |       225187
- | fulfillAvailableAdvancedOrders |       149965 |       217284
+ | fulfillAvailableAdvancedOrders |       149965 |       217284
- | fulfillOrder                   | -12 (-0.01%) |       225067
+ | fulfillOrder                   | -24 (-0.02%) |       225067
```

```
- | matchOrders                          | -12 (-0.01%) | -24 (-0.01%)
+ | matchOrders                          |       158290 | -24 (-0.01%)
- | validate                             |        53206 |        83915
+ | validate                             | -72 (-0.14%) | -48 (-0.06%)
- ========================================================================
- | runtime size                         |        23583 |
+ | runtime size                         | -15 (-0.06%) |
- | init code size                       | +78 (+0.29%) |
+ | init code size                       | +63 (+0.24%) |
========================================================================
```

Added together, the max gas saving counted here is 150.

[G-04] Swap conditions for a better happy path

Estimated savings: 6 gas
Max savings according to `yarn profile` *: 38 gas*

When a staticcall ends in failure, there will rarely, if ever, be a case of `returndatasize()` being non-zero. However, most often with a staticcall, `success` will be true, while the `returndatasize()` has a higher probability of being 0. The consequence is that, in the current order of conditions, both conditions are more likely to be evaluated. Furthermore, the RETURNDATASIZE opcode costs 2 gas while a MLOAD costs 3 gas. Consider swapping both conditions here for a better happy path:

```
File: PointerLibraries.sol
215:         assembly {
216:             let success := staticcall(
217:                 gas(),
218:                 IdentityPrecompileAddress,
219:                 src,
220:                 size,
221:                 dst,
222:                 size
223:             )
- 224:             if or(iszero(success), iszero(returndatasize(
+ 224:             if or(iszero(returndatasize()), iszero(succes
225:                 revert(0, 0)
226:             }
```

```
227:         }
```

yarn profile

```
=================================================================
| method                          |          min |          max |
=================================================================
- | fulfillAdvancedOrder           | +12 (+0.01%) |       225187
+ | fulfillAdvancedOrder           | +12 (+0.01%) |       225187
- | fulfillAvailableAdvancedOrders |       149965 |       217284
+ | fulfillAvailableAdvancedOrders |       149965 |       217284
- | matchOrders                    | -12 (-0.01%) | -24 (-0.01%)
+ | matchOrders                    | -12 (-0.01%) | -24 (-0.01%)
- | validate                       |        53206 |        83915
+ | validate                       |        53206 | -12 (-0.01%)
```

Added together, the max gas saving counted here is 38.

[G-05] Optimized operations

Estimated savings: 3 gas
Max savings according to `yarn profile` *: 58 gas*

Tested on Remix: The optimized equivalent of `or(eq(a, 2), eq(a, 3))` is `and(lt(a, 4),gt(a, 1))` (saving 3 gas)

Proof of Concept

The following opcodes happen for `and(lt(a, 4),gt(a, 1))`:

```
PUSH 4   4
DUP2    lt(a, 4)
LT    lt(a, 4)
PUSH 1   1
SWAP1    gt(a, 1)
SWAP2    gt(a, 1)
GT    gt(a, 1)
AND    and(lt(a, 4), gt(a, 1))
```

```
SWAP1    and(lt(a, 4), gt(a, 1))
```

The following opcodes happen for `or(eq(a, 2), eq(a, 3))`:

```
PUSH 2   2
DUP2     eq(a, 2)
EQ     eq(a, 2)
PUSH 3   3
SWAP2    eq(a, 3)
SWAP1    eq(a, 3)
SWAP2    eq(a, 3)
EQ     eq(a, 3)
OR     or(eq(a, 2), eq(a, 3))
SWAP1    or(eq(a, 2), eq(a, 3))
```

As we can see here, an extra SWAP is costing an extra 3 gas compared to the optimized version.
Consider replacing with the following:

```
File: ZoneInteraction.sol
140:     function _isRestrictedAndCallerNotZone(
141:         OrderType orderType,
142:         address zone
143:     ) internal view returns (bool mustValidate) {
144:         assembly {
145:             mustValidate := and(
- 146:                 or(eq(orderType, 2), eq(orderType, 3)),
+ 146:                 and(lt(orderType, 4),gt(orderType, 1)),
147:                 iszero(eq(caller(), zone))
148:             )
149:         }
150:     }
```

yarn profile

```
=================================================================
| method                          |          min |          max |
=================================================================
```

```
- | cancel                          |        41219 |        58403
+ | cancel                          | +12 (+0.03%) | -12 (-0.02%)
- | fulfillAdvancedOrder            | +12 (+0.01%) |       225187
+ | fulfillAdvancedOrder            |        96287 |       225187
- | fulfillBasicOrder               |        91377 |     -12 (0%)
+ | fulfillBasicOrder               | -24 (-0.03%) |      1621539
- | matchOrders                     | -12 (-0.01%) | -24 (-0.01%)
+ | matchOrders                     | -12 (-0.01%) | -24 (-0.01%)
- | validate                        |        53206 |        83915
+ | validate                        |        53206 |        83915
```

Added together, the max gas saving counted here is 58.

[G-06] Pre-decrements cost less than post-decrements

Estimated savings: 5 gas per iteration
Max savings according to `yarn profile` *: 61 gas*

For a `uint256 maximumFulfilled` variable, the following is true with the Optimizer enabled at 10k:

- `--maximumFulfilled` costs 5 gas less than `maximumFulfilled--`

Affected code:

```
File: OrderCombiner.sol
- 272:                 maximumFulfilled--;
+ 272:                 --maximumFulfilled;
```

yarn profile

```
================================================================
| method                          |          min |          max |
================================================================
- | matchAdvancedOrders             | +12 (+0.01%) |     -12 (0%)
+ | matchAdvancedOrders             | -36 (-0.02%) |     -12 (0%)
- | matchOrders                     | -12 (-0.01%) | -24 (-0.01%)
+ | matchOrders                     | -12 (-0.01%) | -24 (-0.01%)
- | validate                        |        53206 |        83915
```

```
+ | validate                                |        53206 | -12 (-0.01%)
```

Added together, the max gas saving counted here is 61.

0age (OpenSea) commented:

> Lovely optimizations 🏅

hickuphh3 (judge) commented:

> NGL the detail and analysis for number 5 is pretty sick!

0age (OpenSea) resolved:

> [G-01] Using XOR (^) and OR (|) bitwise equivalents:
> https://github.com/ProjectOpenSea/seaport/pull/908

> [G-02] Shift left by 5 instead of multiplying by 32:
> https://github.com/ProjectOpenSea/seaport/pull/909

> [G-03] Using a positive conditional flow to save a NOT opcode:
> https://github.com/ProjectOpenSea/seaport/pull/910

> [G-04] Swap conditions for a better happy path:
> https://github.com/ProjectOpenSea/seaport/pull/912

> [G-05] Optimized operations:
> https://github.com/ProjectOpenSea/seaport/pull/911

> [G-06] Pre-decrements cost less than post-decrements:
> https://github.com/ProjectOpenSea/seaport/pull/913

Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top