LooksRare Aggregator 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 LooksRare Aggregator smart contract system written in Solidity. The audit contest took place between November 8—November 13 2022.

Wardens

78 Wardens contributed reports to the LooksRare Aggregator contest:

1. 0x1f8b
2. 0x52
3. 0xSmartContract
4. 0xc0ffEE
5. 0xhacksmithh
6. 8olidity
7. Awesome
8. Aymen0909
9. BClabs (nalus and Reptilia)
10. Bnke0x0
11. Chom

12. CloudX (Migue, pabliyo and marce1993)
13. Decurity (Beched, me_na0mi and raz0r)
14. Deivitto
15. HE1M
16. Hashlock
17. lllllll
18. Josiah
19. KingNFT
20. Koolex
21. Lambda
22. M0ndoHEHE
23. Nyx
24. R2
25. RaymondFam
26. ReyAdmirado
27. Rolezn
28. SamGMK
29. Sathish9098
30. SinceJuly
31. V_B (Barichek and vlad_bochok)
32. Vadis
33. Waze
34. a12jmx
35. adriro
36. ajtra
37. aphak5010
38. aviggiano
39. bearonbike
40. bin

41. brgltd
42. carlitox477
43. carrotsmuggler
44. cccz
45. ch0bu
46. chaduke
47. corerouter
48. datapunk
49. delfin454000
50. erictee
51. fatherOfBlocks
52. fs0c
53. gianganhnguyen
54. gz627
55. hanxin
56. horsefacts
57. hyh
58. jayphbee
59. joestakey
60. koxuan
61. ktg
62. ladboy233
63. pashov
64. perseverancesuccess
65. rbserver
66. ret2basic
67. ronnyx2017
68. shark
69. teawaterwire

70. tnevler
71. vv7
72. zaskoh

This contest was judged by [Picodes](#).

Final report assembled by [itsmetechjay](#).

Summary

The C4 analysis yielded an aggregated total of 6 unique vulnerabilities. Of these vulnerabilities, 0 received a risk rating in the category of HIGH severity and 6 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 54 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 12 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 LooksRare Aggregator contest repository](#), and is composed of 26 smart contracts written in the Solidity programming language and includes 1,230 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 (6)

[M-01] The `_executeNonAtomicOrders` function in SeaportProxy.sol may fail unexpectedly

Submitted by KingNFT

The `\_executeNonAtomicOrders` function in SeaportProxy.sol tries to send fees by batch. This may break the `NonAtomic` feature.

Proof of Concept

Let’s say the user wants to buy 3 NFTs with the following order parameters:

```
NFT_1: price = 100 USDT, fee = 5%
NFT_2: price = 200 USDT, fee = 5%
NFT_3: price = 300 USDT, fee = 5%
```

Given the user only sends 600 USDT, the expected result should be

```
NFT_1: success
NFT_2: success
NFT_3: failed
```

But as the fees are batched and sent at last, cause all 3 orders failed.

```
function _executeNonAtomicOrders(
    BasicOrder[] calldata orders,
    bytes[] calldata ordersExtraData,
    address recipient,
    uint256 feeBp,
    address feeRecipient
) private {
    uint256 fee;
    address lastOrderCurrency;
    for (uint256 i; i < orders.length; ) {
        OrderExtraData memory orderExtraData = abi.decode(orders
        AdvancedOrder memory advancedOrder;
        advancedOrder.parameters = _populateParameters(orders[i]
        advancedOrder.numerator = orderExtraData.numerator;
        advancedOrder.denominator = orderExtraData.denominator;
        advancedOrder.signature = orders[i].signature;

        address currency = orders[i].currency;
        uint256 price = orders[i].price;

        try
            marketplace.fulfillAdvancedOrder{value: currency ==
                advancedOrder,
                new CriteriaResolver[](0),
                bytes32(0),
                recipient
            )
        {
            if (feeRecipient != address(0)) {
                uint256 orderFee = (price * feeBp) / 10000;
                if (currency == lastOrderCurrency) {
                    fee += orderFee;
                } else {
                    if (fee > 0) _transferFee(fee, lastOrderCurr

                    lastOrderCurrency = currency;
                    fee = orderFee;
                }
            }
        } catch {}

        unchecked {
            ++i;
        }
    }

    if (fee > 0) _transferFee(fee, lastOrderCurrency, feeRecipie
}
```

Tools Used

VS Code

Recommended Mitigation Steps

Don't batch up fees.

[0xhiroshi (LooksRare) disputed and commented](#):

> I would treat this as a malicious user trying to buy NFTs without paying for fees as he should be paying 600 x 1.05 = 630 instead of 600. So I think it's ok to just let it revert.

[KingNFT (warden) commented](#):

> Sorry for not indicating clearly. The reason I think this may happen is that the NFT price may increase during submitting of tx, not due to a malicious user.

[0xhiroshi (LooksRare) acknowledged and commented](#):

> Even if the seller submits a new listing with a higher price, the current transaction should still be valid. When there are two listings for the same NFT, they are both valid unless the seller explicitly cancels the original order on-chain, usually through the cancellation of an order nonce.

[M-02] Too much fee charged when Seaport is partially filled

Submitted by [cccz](#)

When a user fulfills an order using SeaportProxy, fees are charged in the `\_handleFees` function based on orders.price.

```
    function _handleFees(
        BasicOrder[] calldata orders,
        uint256 feeBp,
        address feeRecipient
    ) private {
        address lastOrderCurrency;
        uint256 fee;
        uint256 ordersLength = orders.length;

        for (uint256 i; i < ordersLength; ) {
            address currency = orders[i].currency;
            uint256 orderFee = (orders[i].price * feeBp) / 1000(
```

According to the Seaport documentation, Seaport allows partial fulfillment of orders, which results in too much fee being charged when an order is partially filled
https://docs.opensea.io/v2.0/reference/seaport-overview#partial-fills

Consider feeBp == 2%

The order on Seaport has a fill status of 0/100 and each item is worth 1 eth. User A fulfills the order using LooksRareAggregator.execute and sends 102 ETH, where order.price == 100 ETH.

Since the other user fulfilled the order before User A, when User A fulfills the order, the order status is 99/100.

Eventually User A buys an item for 1 ETH but pays a fee of 2 ETH.

Proof of Concept

https://github.com/code-423n4/2022-11-looksrare/blob/e3b2c053f722b0ca2dce3a3eb06f64859b8b7a6f/contracts/proxies/SeaportProxy.sol#L136-L147

Recommended Mitigation Steps

Consider charging fees based on the user’s actual filled price.

[0xhiroshi (LooksRare) confirmed and commented]():

> We are dropping fees completely.

[M-03] It is clearly stated that timelock is used, but this does not happen in the codes

Submitted by OxSmartContract, also found by carrotsmuggler

It is stated in the documents that "contract ownership management" is used with a timelock;

```
README.md:
  122: - Does it use a timelock function?: Yes but only for cont
```

However, the `function _setupDelayForRenouncingOwnership` where timelock is specified in the `OwnableTwoSteps.sol` contract where `owner` privileges are set is not used in the project, so a timelock cannot be mentioned.

```
 function _setupDelayForRenouncingOwnership(uint256 _delay) inte
        delay = _delay;
    }
```

This is stated in the NatSpec comments but there is no definition as stated in the comments;

```
contracts/OwnableTwoSteps.sol:
  40:      *         Delay (for the timelock) must be set by the
```

Recommended Mitigation Steps

```
contracts/OwnableTwoSteps.sol:

    // Delay for the timelock (in seconds)
    uint256 public delay;

  43       */
  44:     constructor(uint256 _delay) {
  45:         owner = msg.sender;
  +           delay = _delay;
  46:     }
  47:
  48      /**
```

0xhiroshi (LooksRare) confirmed

[M-04] Users can avoid paying any fees when using ERC20EnabledLooksRareAggregator for Seaport

Submitted by ronnyx2017, also found by 0x52

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/proxies/SeaportProxy.sol#L136-L164

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/proxies/SeaportProxy.sol#L232-L252

Impact

The `order.price` in the parameter `tradeData` is not used as the actual token amount sent to the seaport market and also not checked if those are equal when using the `ERC20EnabledLooksRareAggregator` for `SeaportPorxy` with ERC20 tokens.

So users can set the order.price to ZERO to avoid paying any fees for ERC20 orders.

Proof of Concept

Test file SeaportUSDCZeroPrice.t.sol, modified from test SeaportProxyERC721USDC.t.sol and annotate with `# diff`.

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

import {IERC20} from "../../contracts/interfaces/IERC20.sol";
import {IERC721} from "../../contracts/interfaces/IERC721.sol";
import {OwnableTwoSteps} from "../../contracts/OwnableTwoSteps.s
import {SeaportProxy} from "../../contracts/proxies/SeaportProxy
import {ERC20EnabledLooksRareAggregator} from "../../contracts/E
import {LooksRareAggregator} from "../../contracts/LooksRareAggr
import {IProxy} from "../../contracts/interfaces/IProxy.sol";
import {ILooksRareAggregator} from "../../contracts/interfaces/I
import {BasicOrder, FeeData, TokenTransfer} from "../../contract
import {TestHelpers} from "./TestHelpers.sol";
import {TestParameters} from "./TestParameters.sol";
import {SeaportProxyTestHelpers} from "./SeaportProxyTestHelpers

/**
 * @notice SeaportProxy ERC721 USDC orders with fees tests
 */
contract SeaportUSDCZeroPrice is TestParameters, TestHelpers, Se
    LooksRareAggregator private aggregator;
    ERC20EnabledLooksRareAggregator private erc20EnabledAggregat
    SeaportProxy private seaportProxy;

    function setUp() public {
        vm.createSelectFork(vm.rpcUrl("mainnet"), 15_491_323);

        aggregator = new LooksRareAggregator();
        erc20EnabledAggregator = new ERC20EnabledLooksRareAggreg
        seaportProxy = new SeaportProxy(SEAPORT, address(aggrega
        aggregator.addFunction(address(seaportProxy), SeaportPro

        deal(USDC, _buyer, INITIAL_USDC_BALANCE);

        aggregator.approve(SEAPORT, USDC, type(uint256).max);
        aggregator.setFee(address(seaportProxy), 250, _protocolF
        aggregator.setERC20EnabledLooksRareAggregator(address(er
    }

    function testExecuteWithPriceZero() public asPrankedUser(_bu
        bool isAtomic = true;
        ILooksRareAggregator.TradeData[] memory tradeData = _gen
        uint256 totalPrice =
        // diff
        // not pay the fee for order 0 , so cut 250 bp from tota
        (tradeData[0].orders[0].price * (10250 - 250)) /
        // diff end
            10000 +
            (tradeData[0].orders[1].price * 10250) /
            10000;
        IERC20(USDC).approve(address(erc20EnabledAggregator), to
        // diff
        // set order 0 price to ZERO
        tradeData[0].orders[0].price = 0;
        // diff end

        TokenTransfer[] memory tokenTransfers = new TokenTransfe
        tokenTransfers[0].currency = USDC;
        tokenTransfers[0].amount = totalPrice;

        erc20EnabledAggregator.execute(tokenTransfers, tradeData

        assertEq(IERC721(BAYC).balanceOf(_buyer), 2);
        assertEq(IERC721(BAYC).ownerOf(9948), _buyer);
        assertEq(IERC721(BAYC).ownerOf(8350), _buyer);
        assertEq(IERC20(USDC).balanceOf(_buyer), INITIAL_USDC_BA
    }

    function _generateTradeData() private view returns (ILooksRa
        BasicOrder memory orderOne = validBAYCId9948Order();
        BasicOrder memory orderTwo = validBAYCId8350Order();
        BasicOrder[] memory orders = new BasicOrder[](2);
        orders[0] = orderOne;
        orders[1] = orderTwo;

        bytes[] memory ordersExtraData = new bytes[](2);
        {
            bytes memory orderOneExtraData = validBAYCId9948Orde
            bytes memory orderTwoExtraData = validBAYCId8350Orde
            ordersExtraData[0] = orderOneExtraData;
            ordersExtraData[1] = orderTwoExtraData;
        }

        bytes memory extraData = validMultipleItemsSameCollectio
        ILooksRareAggregator.TradeData[] memory tradeData = new
        tradeData[0] = ILooksRareAggregator.TradeData({
            proxy: address(seaportProxy),
            selector: SeaportProxy.execute.selector,
            value: 0,
            maxFeeBp: 250,
            orders: orders,
            ordersExtraData: ordersExtraData,
            extraData: extraData
        });

        return tradeData;
    }
}
```

run test:

```
forge test --match-test testExecuteWithPriceZero -vvvvv
```

Tools Used

Foundry

Recommended Mitigation Steps

Assert the order price is equal to the token amount of the seaport order when populating parameters.

Oxhiroshi (LooksRare) confirmed, but disagreed with severity and commented:

> The worst case is for us to not able to collect fees, and we can technically just deploy a new contract to fix this. Not sure if this is considered assets stolen/lost/compromise?

Oxhiroshi (LooksRare) commented:

> Update: we are dropping fees completely.

Picodes (judge) decreased severity to Medium and commented:

> Medium severity as no user funds are at risk, and the impact for the protocol would be minimal.

[M-05] `call` opcode's return value not checked.

Submitted by jayphbee, *also found by* V_B, Decurity, gz627, joestakey, 0xc0ffEE, corerouter, SinceJuly, carlitox477, rbserver, Vadis, 0x52, chaduke, *and* aviggiano

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/lowLevelCallers/LowLevelETH.sol#L35

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/lowLevelCallers/LowLevelETH.sol#L46 https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/lowLevelCallers/LowLevelETH.sol#L57

Impact

The `call` opcode's return value not checked, which could lead to the `originator` losing funds.

Proof of Concept

The caller of `LooksRareAggregator.sol::execute` could be a contract who may not implement the `fallback` or `receive` function, when a call to it with value sent, it will revert, thus failed to receive the ETH.

Let's imagine the contract calls the `execute` function to buy multiple NFTs with ETH as the payout currency and make the `isAtomic` parameter being false. Since the batch buy of NFTs is not atomic, the failed transactions in LooksRare or Seaport marketplace will return the passed ETH. The contract doesn't implement the `fallback/receive` function and the call opcode's return value not checked, thus the ETH value will be trapped in the `LooksRareAggregator` contract until the next user call the `execute` function and the trapped ETH is returned to him. The `originator` lose funds.

```
    function _returnETHIfAny(address recipient) internal {
        assembly {
            if gt(selfbalance(), 0) {
                let status := call(gas(), recipient, selfbalance
            }
        }
    }
```

Recommended Mitigation Steps

Check the return value on the `call` opcode.

```
    function _returnETHIfAny() internal {
        bool status;
        assembly {
            if gt(selfbalance(), 0) {
                status := call(gas(), caller(), selfbalance(), (
            }
        }
        if (!status) revert ETHTransferFail();
    }

    function _returnETHIfAny(address recipient) internal {
        bool status;
        assembly {
            if gt(selfbalance(), 0) {
                status := call(gas(), recipient, selfbalance(),
            }
        }
        if (!status) revert ETHTransferFail();
    }
function _returnETHIfAnyWithOneWeiLeft() internal {
        bool status;
        assembly {
            if gt(selfbalance(), 1) {
                status := call(gas(), caller(), sub(selfbalance(
            }
        }
        if (!status) revert ETHTransferFail();
    }
```

[0xhiroshi (LooksRare) confirmed]

[Picodes (judge) decreased severity to Medium and commented]:

> Medium severity as only the dust is impacted.

[M-06] Public to all funds escape

Submitted by V_B, *also found by* Decurity, hyh, carrotsmuggler, vv7, teawaterwire, M0ndoHEHE, hanxin, HE1M, rbserver, R2, zaskoh, Vadis, koxuan, Koolex, aviggiano, chaduke, *and* Rolezn

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/TokenRescuer.sol#L22 https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/TokenRescuer.sol#L34

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/LooksRareAggregator.sol#L27

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/LooksRareAggregator.sol#L108

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/LooksRareAggregator.sol#L109

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/LooksRareAggregator.sol#L245

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/lowLevelCallers/LowLevelETH.sol#L43

Vulnerability Details

The `LooksRareAggregator` smart contract implements a bunch of functions to escape funds by the contract owner (see `rescueETH`, `rescueERC20`, `rescueERC721`, and `rescueERC1155`). In this way, any funds that were accidentally sent to the contract or were locked due to incorrect contract implementation can be returned to the owner. However, locked funds can be rescued by anyone without the owner's permission. This is completely contrary to the idea of having rescue functions.

In order to withdraw funds from the contract, a user may just call the `execute` function in the `ERC20EnabledLooksRareAggregator` with `tokenTransfers` that contains the addresses of tokens to be withdrawn.

Thus, after the order execution `_returnERC20TokensIfAny` and `_returnETHIfAny` will be called, and the whole balance of provided ERC20 tokens and Ether will be returned to `msg.sender`.

Please note, that means that the owner can be front-ran with `rescue` functions and an attacker will receive funds instead.

Impact

Useless of rescue functionality and vulnerability to jamming funds.

Recommended Mitigation Steps

`_returnETHIfAny` and `_returnERC20TokensIfAny` should return the amount of the token that was deposited.

Picodes (judge) decreased severity to Medium and commented:

> As only stuck funds are at risk, and as the aggregator contract itself is not supposed to handle funds, I don't think this qualifies for High Severity.

0xhiroshi (LooksRare) disputed and commented:

> We have decided that any ERC20 tokens sent there accidentally are free for all.

Picodes (judge) commented:

> Keeping the Medium severity because the contract implements `TokenRescuer`, so the intent "that any ERC20 tokens sent there accidentally are free for all" totally makes sense but wasn't clear prior to the audit. So I consider this a case where tokens that should belong to the protocol could be withdrawn by anyone.

Low Risk and Non-Critical Issues

For this contest, 53 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by **RaymondFam** received the top score from the judge.

The following wardens also submitted reports: V_B, adriro, ajtra, Josiah, horsefacts, Chom, fs0c, rbserver, tnevler, Waze, ch0bu, brgltd, jayphbee, a12jmx, 0xc0ffEE, bearonbike, delfin454000, pashov, carrotsmuggler, ret2basic, Deivitto, 0xhacksmithh, 0xSmartContract, Sathish9098, IllIIII, SinceJuly, carlitox477, R2, Awesome, erictee, ReyAdmirado, Rolezn, Vadis, Hashlock, ktg, zaskoh, 0x52, ladboy233, cccz, Nyx, Bnke0x0, BClabs, SamGMK, KingNFT, perseverancesuccess, fatherOfBlocks, aphak5010, datapunk, bin, 0x1f8b, 8olidity, *and* chaduke .

[01] Un-indexed Parameters in Events

Consider indexing parameters for events, serving as logs filter when looking for specifically wanted data. Up to three parameters in an event function can receive the attribute `indexed` which will cause the respective arguments to be treated as log topics instead of data. Here is one of the instances entailed:

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/interfaces/ILooksRareAggregator.sol#L44

```
    event FeeUpdated(address proxy, uint256 bp, address recipier
```

[02] Descriptive and Verbose Options

Consider making the naming of local variables more verbose and descriptive so all other peer developers would better be able to comprehend the intended statement logic, significantly enhancing the code readability. Here is one of the instances entailed:

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/interfaces/ILooksRareAggregator.sol#L44

```
    @ bp
    event FeeUpdated(address proxy, uint256 bp, address recipier
```

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/libraries/OrderStructs.sol#L25

```
    @ bp
    uint256 bp; // Aggregator fee basis point
```

[03] Typo Mistakes

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/libraries/seaport/ConsiderationStructs.sol#L168

```
    @ are
 *      type is restricted and the offerer or zone are not the c
```

[04] Sanity/Threshold/Limit Checks

Devoid of sanity/threshold/limit checks, critical parameters can be configured to invalid values, causing a variety of issues and breaking expected interactions within/between contracts. Consider adding proper `uint256` validation as well as zero address checks in the constructor. A worst case scenario would render the contract needing to be re-deployed in the event of human/accidental errors that involve value assignments to immutable variables. If the validation procedure is unclear or too complex to implement on-chain, document the potential issues that could produce invalid values.

Consider also adding an optional codehash check for immutable address at the constructor since the zero address check cannot guarantee a matching address has been inputted.

These checks are generally not implemented in the contracts. As an example, the following constructor may be refactored to:

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/proxies/LooksRareProxy.sol#L37-L40

```
    constructor(address _marketplace, address _aggregator, bytes
        if (_marketplace == address(0) || _aggregator == address
            _marketplace.codehash != _marketplaceCodeHash || _ag
            revert InvalidAddress();
        }
        marketplace = ILooksRareExchange(_marketplace);
        aggregator = _aggregator;
    }
```

All other instances entailed:

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/proxies/SeaportProxy.sol#L45-L48
https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/ERC20EnabledLooksRareAggregator.sol#L21-L23

[05] Empty/Unused Function Parameters

Empty or unused function parameters should be commented out as a better and declarative way to silence runtime warning messages. As an example, the following function may have these parameters refactored to:

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/proxies/LooksRareProxy.sol#L50-L58

```
    function execute(
        BasicOrder[] calldata orders,
        bytes[] calldata ordersExtraData,
        bytes memory /* extraData */,
        address recipient,
        bool isAtomic,
        uint256 /* feeBp */,
        address /* feeRecipient */
    ) external payable override {
```

[06] Inadequate NatSpec

Solidity contracts can use a special form of comments, i.e., the Ethereum Natural Language Specification Format (NatSpec) to provide rich documentation for functions, return variables and more. Please visit the following link for further details:

https://docs.soliditylang.org/en/v0.8.16/natspec-format.html

Here are some of the instances entailed:

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/proxies/LooksRareProxy.sol#L107-L134

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/proxies/SeaportProxy.sol#L88-L269 https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/LooksRareAggregator.sol#L51-L112

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/LooksRareAggregator.sol#L220-L251

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/TokenReceiver.sol

[07] Use of `ecrecover` is Susceptible to Signature Malleability

The built-in EVM pre-compiled `ecrecover` featured in `SignatureChecker.sol` (https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/SignatureChecker.sol) is susceptible to signature malleability due to non-unique v and s (which may not always be in the lower half of the modulo set) values, possibly leading to replay attacks. Elsewhere if devoid of the adoption of nonces, this could prove a vulnerability when not carefully used.

Consider using OpenZeppelin’s ECDSA library which has been time tested in preventing this malleability where possible.

[08] Lines Too Long

Lines in source code are typically limited to 80 characters, but it’s reasonable to stretch beyond this limit when need be as monitor screens theses days are comparatively larger. Considering the files will most likely reside in GitHub that will have a scroll bar automatically kick in when the length is over 164 characters, all code lines and comments should be split when/before hitting this length. Keep line width to max 120 characters for better readability where possible. Here are some of the instances entailed:

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/proxies/SeaportProxy.sol#L35

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/proxies/SeaportProxy.sol#L37

[09] Function Calls in Loop Could Lead to Denial of Service

Function calls made in unbounded loop are error-prone with potential resource exhaustion as it can trap the contract due to the gas limitations or failed transactions. Here are some of the instances entailed:

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/proxies/SeaportProxy.sol#L102

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/proxies/SeaportProxy.sol#L126

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/proxies/SeaportProxy.sol#L145

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/proxies/SeaportProxy.sol#L187

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/proxies/SeaportProxy.sol#L255

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/ERC20EnabledLooksRareAggregator.sol#L41

Consider bounding the loop where possible to avoid unnecessary gas wastage and denial of service.

[10] Failed Function Call Could Occur Without Contract Existence Check

Performing a low-level calls without confirming contract's existence (not yet deployed or have been destructed) could return success even though no function call was executed. Here are the four contract instances entailed:

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/lowLevelCallers/LowLevelETH.sol

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/lowLevelCallers/LowLevelERC20Transfer.sol https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/lowLevelCallers/LowLevelERC721Transfer.sol

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/lowLevelCallers/LowLevelERC1155Transfer.sol

[11] Not Completely Using OpenZeppelin Contracts

OpenZeppelin maintains a library of standard, audited, community-reviewed, and battle-tested smart contracts. Instead of always importing these contracts, the protocol project re-implements them in some cases. This increases the amount of code that the protocol team will have to maintain and miss all the improvements and bug fixes that the OpenZeppelin team is constantly implementing with the help of the community.

Consider importing the OpenZeppelin contracts instead of re-implementing or copying them. These contracts can be extended to add the extra functionalities required if need be.

Here is one of the instances entailed:

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/ReentrancyGuard.sol#L6-L10

```
/**
 * @title ReentrancyGuard
 * @notice This contract protects against reentrancy attacks.
 *         It is adjusted from OpenZeppelin.
 */
```

[12] Events Associated With Setter Functions

Consider having events associated with setter functions emit both the new and old values instead of just the new value. Here is one of the instances entailed:

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/LooksRareAggregator.sol#L162

```
        emit FeeUpdated(proxy, bp, recipient);
```

[13] Add a Timelock to Critical Parameter Change

It is a good practice to give time for users to react and adjust to critical changes with a mandatory time window between them. The first step merely broadcasts to users that a particular change is coming, and the second step commits that change after a suitable waiting period. This allows users that do not accept the change to withdraw within the grace period. A timelock provides more guarantees and reduces the level of trust required, thus decreasing risk for users. It also indicates that the project is legitimate (less risk of a malicious Owner making any malicious or ulterior intention). Specifically, privileged roles could use front running to make malicious changes just ahead of incoming transactions, or purely accidental negative effects could occur due to the unfortunate timing of changes.

Consider extending the timelock feature beyond contract ownership management to business critical functions. Here are some of the instances entailed:

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/LooksRareAggregator.sol#L132

```
    function addFunction(address proxy, bytes4 selector) externa
```

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/LooksRareAggregator.sol#L143

```
    function removeFunction(address proxy, bytes4 selector) exte
```

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/LooksRareAggregator.sol#L153-L157

```
    function setFee(
        address proxy,
        uint256 bp,
        address recipient
        ) external onlyOwner {
```

[14] Contract Owner Has Too Many Privileges

The owner of the contracts has too many privileges relative to standard users. The consequence is disastrous if the contract owner's private key has been compromised. And, in the event the key was lost or unrecoverable, no implementation upgrades and system parameter updates will ever be possible.

For a project this grand, it increases the likelihood that the owner will be targeted by an attacker, especially given the insufficient protection on sensitive owner private keys. The concentration of privileges creates a single point of failure; and, here are some of the incidents that could possibly transpire:

Transfer ownership and mess up with all the setter functions, hijacking the entire protocol.

Consider:

1. splitting privileges (e.g. via the multisig option) to ensure that no one address has excessive ownership of the system,
2. clearly documenting the functions and implementations the owner can change,
3. documenting the risks associated with privileged users and single points of failure, and
4. ensuring that users are aware of all the risks associated with the system.

[15] Added Measures When Calling Seaport Functions

A high risk edge case bug associated with the Seaport `_validateOrderAndUpdateStatus()` was found in the [May code4rena audit contest.]() It concerns truncation to zero on both the numerator and the denominator particularly when involving a restricted token sale. The mitigation steps recommended was not deemed ideal then, and although the Seaport protocol team has since fixed this bug with added GCD measure and removing `unchecked {...}`, it is recommended adding the complementary check to circumvent any other hidden issues associated with it whilst having the error detected at its earliest possibility.

For instance, instead of allowing user to input `2**118` and `2**119` as numerator and denominator for an intended fraction of `1/2`, stem it by making sure that those two inputs could not be more than the total ERC721/1155 collection availability and multiply them with factor just enough to remove the decimals. Here are the two for loop instances entailed prior to calling `marketplace.fulfillAvailableAdvancedOrders()` and `marketplace.fulfillAdvancedOrder()`:

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/proxies/SeaportProxy.sol#L102-L114

https://github.com/code-423n4/2022-11-looksrare/blob/main/contracts/proxies/SeaportProxy.sol#L187-L193

0xhiroshi (LooksRare) commented:

> Un-indexed Parameters in Events - We are going to use Subgraph and our own indexer to index events. We are also not likely going to be able to retrieve events after EIP-4444 passes.

> Descriptive and Verbose Options - Won't fix, already explained in Natspec

> Typo Mistakes - Won't fix, copied from Seaport

> Sanity/Threshold/Limit Checks - Invalid, sounds like overengineering to me

> Empty/Unused Function Parameters - Valid

> Inadequate NatSpec - We only document public/external functions

> Use of ecrecover is Susceptible to Signature Malleability - We don't use the signature as the key for tracking.

> Lines Too Long - Valid

> Function Calls in Loop Could Lead to Denial of Service - Valid

> Not Completely Using OpenZeppelin Contracts - Invalid, this is intentional

> Events Associated With Setter Functions - Invalid, while it's more convenient to have both values, it's not necessary.

> Add a Timelock to Critical Parameter Change - Invalid for fees, there is already fee slippage protection. For add/remove function, the worst case is for a user's transaction to be reverted and lose some gas fees.

> Contract Owner Has Too Many Privileges - Our contracts are owned by multi-sig, while it's not absolutely secure, it's also not just one private key. Won't fix.

> Added Measures When Calling Seaport Functions - Won't fix, we pass the responsibility of this validation to Seaport, it should revert if a user passes malicious nominator/denominator.

Gas Optimizations

For this contest, 12 reports were submitted by wardens detailing gas optimizations. The report highlighted below by IllIllI received the top score from the judge.

The following wardens also submitted reports: tnevler, Aymen0909, CloudX, carlitox477, shark, datapunk, Rolezn, zaskoh, aviggiano, gianganhnguyen, *and* 0x1f8b .

Gas Optimizations Summary

	Issue	Instances	Total Gas Saved
[G-01]	Multiple `address`/ID mappings can be combined into a single `mapping` of an `address`/ID to a `struct`, where appropriate	1	-
[G-02]	Using `calldata` instead of `memory` for read-only arguments in `external` functions saves gas	3	360
[G-03]	State variables should be cached in stack variables rather than re-reading them from storage	2	194
[G-04]	Optimize names to save gas	8	176
[G-05]	`internal` functions not called by the contract should be removed to save deployment gas	2	-

	Issue	Instances	Total Gas Saved
[G-06]	Functions guaranteed to revert when called by normal users can be marked `payable`	13	273

Total: 29 instances over 6 issues with **1003 gas** saved

Gas totals use lower bounds of ranges and count two iterations of each `for` -loop. All values above are runtime, not deployment, values; deployment values are listed in the individual issue descriptions. The table above as well as its gas numbers do not include any of the excluded findings.

[G-01] Multiple `address` /ID mappings can be combined into a single `mapping` of an `address` /ID to a `struct` , where appropriate

Saves a storage slot for the mapping. Depending on the circumstances and sizes of types, can avoid a Gsset (**20000 gas**) per mapping combined. Reads and subsequent writes can also be cheaper when a function requires both values and they both fit in the same storage slot. Finally, if both fields are accessed in the same function, can save **~42 gas per access** due to [not having to recalculate the key's keccak256 hash]() (Gkeccak256 - 30 gas) and that calculation's associated stack operations.

There is 1 instance of this issue:

```
File: contracts/LooksRareAggregator.sol

45        mapping(address => mapping(bytes4 => bool)) private _p
46:       mapping(address => FeeData) private _proxyFeeData;
```

[https://github.com/code-423n4/2022-11-looksrare/blob/e3b2c053f722b0ca2dce3a3eb06f64859b8b7a6f/contracts/LooksRareAggregator.sol#L45-L46]()

```
diff --git a/contracts/LooksRareAggregator.sol b/contracts/Looks
index f215f39..a5525aa 100644
```

```
--- a/contracts/LooksRareAggregator.sol
+++ b/contracts/LooksRareAggregator.sol
@@ -156,8 +156,10 @@ contract LooksRareAggregator is
         address recipient
     ) external onlyOwner {
         if (bp > 10000) revert FeeTooHigh();
-        _proxyFeeData[proxy].bp = bp;
-        _proxyFeeData[proxy].recipient = recipient;
+
+        FeeData storage pfd = _proxyFeeData[proxy];
+        pfd.bp = bp;
+        pfd.recipient = recipient;

         emit FeeUpdated(proxy, bp, recipient);
     }
```

Note that the numbers below are wrong due to this forge bug, where forge doesn't properly track warm/cold slots in tests

```
diff --git a/tmp/gas_before b/tmp/gas_after
index a3222f9..705e068 100644
--- a/tmp/gas_before
+++ b/tmp/gas_after
@@ -17 +17 @@
-│ 2504484
+│ 2504891
@@ -47 +47 @@
-│ setFee
+│ setFee
@@ -230 +230 @@
-│ 13089423
+│ 13089823
@@ -346,0 +347 @@
+Overall gas change: 4929 (0.099%)
```

[G-02] Using `calldata` instead of `memory` for read-only arguments in `external` functions saves gas

When a function with a `memory` array is called externally, the `abi.decode()` step has to use a for-loop to copy each index of the `calldata` to the `memory` index. **Each iteration of this for-loop costs at least 60 gas** (i.e. `60 *`

`<mem_array>.length`). Using `calldata` directly, obliviates the need for such a loop in the contract code and runtime execution. Note that even if an interface defines a function as having `memory` arguments, it's still valid for implementation contracs to use `calldata` arguments instead.

If the array is passed to an `internal` function which passes the array to another internal function where the array is modified and therefore `memory` is used in the `external` call, it's still more gass-efficient to use `calldata` when the `external` function uses modifiers, since the modifiers may prevent the internal functions from being called. Structs have the same overhead as an array of length one

Note that I've also flagged instances where the function is `public` but can be marked as `external` since it's not called by the contract, and cases where a constructor is involved

There are 3 instances of this issue:

```
File: contracts/proxies/LooksRareProxy.sol

50        function execute(
51            BasicOrder[] calldata orders,
52            bytes[] calldata ordersExtraData,
53            bytes memory,
54            address recipient,
55            bool isAtomic,
56            uint256,
57:           address
```

https://github.com/code-423n4/2022-11-looksrare/blob/e3b2c053f722b0ca2dce3a3eb06f64859b8b7a6f/contracts/proxies/LooksRareProxy.sol#L50-L57

```
File: contracts/TokenReceiver.sol

14        function onERC1155Received(
15            address,
16            address,
17            uint256,
18              uint256,
19              bytes memory
20:         ) external virtual returns (bytes4) {

24          function onERC1155BatchReceived(
25              address,
26              address,
27              uint256[] memory,
28              uint256[] memory,
29              bytes memory
30:         ) external virtual returns (bytes4) {
```

https://github.com/code-423n4/2022-11-looksrare/blob/e3b2c053f722b0ca2dce3a3eb06f64859b8b7a6f/contracts/TokenReceiver.sol#L14-L20

```diff
diff --git a/contracts/TokenReceiver.sol b/contracts/TokenRecei
index a82e815..759d4ac 100644
--- a/contracts/TokenReceiver.sol
+++ b/contracts/TokenReceiver.sol
@@ -16,7 +16,7 @@ abstract contract TokenReceiver {
         address,
         uint256,
         uint256,
-        bytes memory
+        bytes calldata
     ) external virtual returns (bytes4) {
         return this.onERC1155Received.selector;
     }
@@ -24,9 +24,9 @@ abstract contract TokenReceiver {
     function onERC1155BatchReceived(
         address,
         address,
-        uint256[] memory,
-        uint256[] memory,
-        bytes memory
+        uint256[] calldata,
+        uint256[] calldata,
+        bytes calldata
     ) external virtual returns (bytes4) {
         return this.onERC1155BatchReceived.selector;
     }
diff --git a/contracts/proxies/LooksRareProxy.sol b/contracts/pr
```

```diff
index cbce118..1ebe936 100644
--- a/contracts/proxies/LooksRareProxy.sol
+++ b/contracts/proxies/LooksRareProxy.sol
@@ -50,7 +50,7 @@ contract LooksRareProxy is IProxy, TokenRescue
     function execute(
         BasicOrder[] calldata orders,
         bytes[] calldata ordersExtraData,
-        bytes memory,
+        bytes calldata,
         address recipient,
         bool isAtomic,
         uint256,
```

```diff
diff --git a/tmp/gas_before b/tmp/gas_after
index a3222f9..94f3986 100644
--- a/tmp/gas_before
+++ b/tmp/gas_after
@@ -17 +17 @@
-| 2504484
+| 2420786
@@ -27 +27 @@
-| execute
+| execute
@@ -31 +31 @@
-| onERC1155Received
+| onERC1155Received
@@ -56 +56 @@
-| 1402373                                                          ¦
+| 1321684                                                          ¦
@@ -62 +62 @@
-| execute                                                          ¦
+| execute                                                          ¦
@@ -71 +71 @@
-| 1683078
+| 1699292
@@ -75 +75 @@
-| execute
+| execute
@@ -230 +230 @@
-| 13089423
+| 12924886
@@ -288 +288 @@
-| mint                                                          ¦  26
+| mint                                                          ¦  26
```

```
@@ -346,0 +347 @@
+Overall gas change: -1174338 (-24.848%)
```

[G-03] State variables should be cached in stack variables rather than re-reading them from storage

The instances below point to the second+ access of a state variable within a function. Caching of a state variable replaces each Gwarmaccess (**100 gas**) with a much cheaper stack read. Other less obvious fixes/optimizations include having local memory caches of state variable structs, or having local caches of state variable contracts/addresses.

There are 2 instances of this issue:

```
File: contracts/OwnableTwoSteps.sol

/// @audit owner on line 87
91:            emit NewOwner(owner);

/// @audit earliestOwnershipRenouncementTime on line 114
116:           emit InitiateOwnershipRenouncement(earliestOwnersh
```

https://github.com/code-423n4/2022-11-looksrare/blob/e3b2c053f722b0ca2dce3a3eb06f64859b8b7a6f/contracts/OwnableTwoSteps.sol#L91

```
diff --git a/contracts/OwnableTwoSteps.sol b/contracts/OwnableTw
index 99d8a7d..366bd03 100644
--- a/contracts/OwnableTwoSteps.sol
+++ b/contracts/OwnableTwoSteps.sol
@@ -84,11 +84,10 @@ abstract contract OwnableTwoSteps is IOwnabl
         if (ownershipStatus != Status.TransferInProgress) rever
         if (msg.sender != potentialOwner) revert WrongPotential

-        owner = msg.sender;
         delete ownershipStatus;
         delete potentialOwner;

-        emit NewOwner(owner);
```

```
+         emit NewOwner(owner = msg.sender);
      }

      /**
@@ -111,9 +110,8 @@ abstract contract OwnableTwoSteps is IOwnabl
          if (ownershipStatus != Status.NoOngoingTransfer) revert

          ownershipStatus = Status.RenouncementInProgress;
-         earliestOwnershipRenouncementTime = block.timestamp + c

-         emit InitiateOwnershipRenouncement(earliestOwnershipRen
+         emit InitiateOwnershipRenouncement(earliestOwnershipRen
      }

      /**
```

Note that the numbers below are wrong due to this forge bug, where forge doesn't properly track warm/cold slots in tests

```
diff --git a/tmp/gas_before b/tmp/gas_after
index a3222f9..f393f3c 100644
--- a/tmp/gas_before
+++ b/tmp/gas_after
@@ -174 +174 @@
-| confirmOwnershipTransfer
+| confirmOwnershipTransfer
@@ -180 +180 @@
-| initiateOwnershipRenouncement
+| initiateOwnershipRenouncement
@@ -346,0 +347 @@
+Overall gas change: 7 (0.008%)
```

[G-04] Optimize names to save gas

`public` / `external` function names and `public` member variable names can be optimized to save gas. See this link for an example of how it works. Below are the interfaces/abstract contracts that can be optimized so that the most frequently-called functions use the least amount of gas possible during method lookup. Method IDs that have two leading zero bytes can save **128 gas** each during deployment, and renaming functions to have lower method IDs will save **22 gas** per call, per sorted position shifted

There are 8 instances of this issue:

```
File: contracts/ERC20EnabledLooksRareAggregator.sol

/// @audit execute()
15:   contract ERC20EnabledLooksRareAggregator is IERC20EnabledI
```

https://github.com/code-423n4/2022-11-looksrare/blob/e3b2c053f722b0ca2dce3a3eb06f64859b8b7a6f/contracts/ERC20EnabledLooksRareAggregator.sol#L15

```
File: contracts/interfaces/IERC20EnabledLooksRareAggregator.sol

/// @audit execute()
7:    interface IERC20EnabledLooksRareAggregator {
```

https://github.com/code-423n4/2022-11-looksrare/blob/e3b2c053f722b0ca2dce3a3eb06f64859b8b7a6f/contracts/interfaces/IERC20EnabledLooksRareAggregator.sol#L7

```
File: contracts/interfaces/ILooksRareAggregator.sol

/// @audit execute()
6:    interface ILooksRareAggregator {
```

https://github.com/code-423n4/2022-11-looksrare/blob/e3b2c053f722b0ca2dce3a3eb06f64859b8b7a6f/contracts/interfaces/ILooksRareAggregator.sol#L6

```
File: contracts/interfaces/IProxy.sol

/// @audit execute()
6:    interface IProxy {
```

https://github.com/code-423n4/2022-11-looksrare/blob/e3b2c053f722b0ca2dce3a3eb06f64859b8b7a6f/contracts/interf aces/IProxy.sol#L6

```
File: contracts/interfaces/SeaportInterface.sol

/// @audit fulfillBasicOrder(), fulfillOrder(), fulfillAdvancedO
28:   interface SeaportInterface {
```

https://github.com/code-423n4/2022-11-looksrare/blob/e3b2c053f722b0ca2dce3a3eb06f64859b8b7a6f/contracts/interfaces/SeaportInterface.sol#L28

```
File: contracts/LooksRareAggregator.sol

/// @audit execute(), setERC20EnabledLooksRareAggregator(), addF
25:   contract LooksRareAggregator is
```

https://github.com/code-423n4/2022-11-looksrare/blob/e3b2c053f722b0ca2dce3a3eb06f64859b8b7a6f/contracts/LooksRareAggregator.sol#L25

```
File: contracts/OwnableTwoSteps.sol

/// @audit cancelOwnershipTransfer(), confirmOwnershipRenounceme
11:   abstract contract OwnableTwoSteps is IOwnableTwoSteps {
```

https://github.com/code-423n4/2022-11-looksrare/blob/e3b2c053f722b0ca2dce3a3eb06f64859b8b7a6f/contracts/OwnableTwoSteps.sol#L11

```
File: contracts/TokenRescuer.sol

/// @audit rescueETH(), rescueERC20()
14:   contract TokenRescuer is OwnableTwoSteps, LowLevelETH, Low
```

https://github.com/code-423n4/2022-11-looksrare/blob/e3b2c053f722b0ca2dce3a3eb06f64859b8b7a6f/contracts/Token

Rescuer.sol#L14

[G-05] `internal` functions not called by the contract should be removed to save deployment gas

If the functions are required by an interface, the contract should inherit from that interface and use the `override` keyword.

There are 2 instances of this issue:

```
File: contracts/lowLevelCallers/LowLevelERC1155Transfer.sol

23        function _executeERC1155SafeTransferFrom(
24            address collection,
25            address from,
26            address to,
27            uint256 tokenId,
28:           uint256 amount
```

https://github.com/code-423n4/2022-11-looksrare/blob/e3b2c053f722b0ca2dce3a3eb06f64859b8b7a6f/contracts/lowLevelCallers/LowLevelERC1155Transfer.sol#L23-L28

```
File: contracts/lowLevelCallers/LowLevelETH.sol

54        function _returnETHIfAnyWithOneWeiLeft() internal {
55:           assembly {
```

https://github.com/code-423n4/2022-11-looksrare/blob/e3b2c053f722b0ca2dce3a3eb06f64859b8b7a6f/contracts/lowLevelCallers/LowLevelETH.sol#L54-L55

[G-06] Functions guaranteed to revert when called by normal users can be marked `payable`

If a function modifier such as `onlyOwner` is used, the function will revert if a normal user tries to pay the function. Marking the function as `payable` will lower the gas cost for legitimate callers because the compiler will not include checks for whether a payment was provided. The extra opcodes avoided are `CALLVALUE` (2), `DUP1` (3), `ISZERO` (3), `PUSH2` (3), `JUMPI` (10), `PUSH1` (3), `DUP1` (3), `REVERT` (0), `JUMPDEST` (1), `POP` (2), which costs an average of about **21 gas per call** to the function, in addition to the extra deployment cost.

There are 13 instances of this issue:

```
File: contracts/LooksRareAggregator.sol

120:      function setERC20EnabledLooksRareAggregator(address _

132:      function addFunction(address proxy, bytes4 selector)

143:      function removeFunction(address proxy, bytes4 selector

153       function setFee(
154           address proxy,
155           uint256 bp,
156           address recipient
157:      ) external onlyOwner {

173       function approve(
174           address marketplace,
175           address currency,
176           uint256 amount
177:      ) external onlyOwner {

197       function rescueERC721(
198           address collection,
199           address to,
200           uint256 tokenId
201:      ) external onlyOwner {

213       function rescueERC1155(
214           address collection,
215           address to,
216           uint256[] calldata tokenIds,
217           uint256[] calldata amounts
218:      ) external onlyOwner {
```

https://github.com/code-423n4/2022-11-looksrare/blob/e3b2c053f722b0ca2dce3a3eb06f64859b8b7a6f/contracts/LooksRareAggregator.sol#L120

```
File: contracts/OwnableTwoSteps.sol

51:        function cancelOwnershipTransfer() external onlyOwner

68:        function confirmOwnershipRenouncement() external onlyO

98:        function initiateOwnershipTransfer(address newPotentia

110:       function initiateOwnershipRenouncement() external only
```

https://github.com/code-423n4/2022-11-looksrare/blob/e3b2c053f722b0ca2dce3a3eb06f64859b8b7a6f/contracts/OwnableTwoSteps.sol#L51

```
File: contracts/TokenRescuer.sol

22:        function rescueETH(address to) external onlyOwner {

34:        function rescueERC20(address currency, address to) ext
```

https://github.com/code-423n4/2022-11-looksrare/blob/e3b2c053f722b0ca2dce3a3eb06f64859b8b7a6f/contracts/TokenRescuer.sol#L22

Excluded Gas Optimization Findings

These findings are excluded from awards calculations because there are publicly-available automated tools that find them. The valid ones appear here for completeness.

	Issue	Instances	Total Gas Saved
[G-07]	`<array>.length` should not be looked up in every loop of a `for` -loop	2	6
[G-08]	Using `bool` s for storage incurs overhead	1	17100

Total: 3 instances over 2 issues with **17106 gas** saved

Gas totals use lower bounds of ranges and count two iterations of each `for` -loop. All values above are runtime, not deployment, values; deployment values are listed in the individual issue descriptions.

[G-07] `<array>.length` should not be looked up in every loop of a `for` -loop

The overheads outlined below are *PER LOOP*, excluding the first loop

- storage arrays incur a Gwarmaccess (**100 gas**)
- memory arrays use `MLOAD` (**3 gas**)
- calldata arrays use `CALLDATALOAD` (**3 gas**)

Caching the length changes each of these to a `DUP<N>` (**3 gas**), and gets rid of the extra `DUP<N>` needed to store the stack offset

There are 2 instances of this issue:

```
File: contracts/proxies/SeaportProxy.sol

/// @audit (valid but excluded finding)
126:            for (uint256 i; i < availableOrders.length; ) {

/// @audit (valid but excluded finding)
187:            for (uint256 i; i < orders.length; ) {
```

https://github.com/code-423n4/2022-11-looksrare/blob/e3b2c053f722b0ca2dce3a3eb06f64859b8b7a6f/contracts/proxi es/SeaportProxy.sol#L126

[G-08] Using `bool` s for storage incurs overhead

```
    // Booleans are more expensive than uint256 or any type that
    // word because each write operation emits an extra SLOAD to
    // slot's contents, replace the bits taken up by the boolear
    // back. This is the compiler's defense against contract upg
    // pointer aliasing, and it cannot be disabled.
```

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/58f635312aa21f947cae5f8578638a85aa2519f5/contracts/security/ReentrancyGuard.sol#L23-L27 Use `uint256(1)` and `uint256(2)` for true/false to avoid a Gwarmaccess (100 gas) for the extra SLOAD, and to avoid Gsset (**20000 gas**) when changing from `false` to `true`, after having been `true` in the past

There is 1 instance of this issue:

```
File: contracts/LooksRareAggregator.sol

/// @audit (valid but excluded finding)
45:       mapping(address => mapping(bytes4 => bool)) private _p
```

https://github.com/code-423n4/2022-11-looksrare/blob/e3b2c053f722b0ca2dce3a3eb06f64859b8b7a6f/contracts/LooksRareAggregator.sol#L45

0xhiroshi (LooksRare) commented:

> G-01 addressed in other issues

> G-02 valid

> G-03 valid

> G-04 invalid

> G-05 invalid - it’s a generalized lib for multiple projects

> G-06 invalid - addressed in other issues

[Picodes (judge) decreased severity to Medium and commented]:

> All findings are theoretically valid in my opinion, although it would not make sense to implement G-05, and G-04 and G-06 provide very little value to the sponsor, if any.

Disclosures