# Blur Exchange contest Findings & Analysis Report

2022-12-08

## 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 Blur Exchange smart contract system written in Solidity. The audit contest took place between October 5—October 10 2022.

### Wardens

87 Wardens contributed reports to the Blur Exchange contest:

1. Soosh
2. 0x4non
3. d3e4
4. rotcivegaf
5. Trust
6. 0x1f8b
7. arcoun
8. RustyRabbit
9. Ruhum

10. saian
11. 0xc0ffEE
12. minhtrng
13. RaymondFam
14. sakshamguruji
15. cccz
16. lllllll
17. Aymen0909
18. ladboy233
19. nicobevi
20. brgltd
21. Rolezn
22. enckrish
23. Lambda
24. csanuragjain
25. exd0tpy
26. zzykxx
27. dipp
28. rvierdiiev
29. 0x52
30. 0xRobocop
31. 8olidity
32. aviggiano
33. bardamu
34. Ch_301
35. cryptonue
36. hansfriese
37. jayphbee
38. Jeiwan

39. joestakey
40. Junnon
41. KIntern_NA (TrungOre and duc)
42. M4TZ1P (DekaiHako, holyhansss_kr, Zer0Luck, AAIIWITF, and exd0tpy)
43. MiloTruck
44. minhquanym
45. Nyx
46. obront
47. PaludoX0
48. polymorphism
49. rokinot
50. romand
51. serial-coder
52. TomJ
53. trustindistrust
54. 0xNazgul
55. 0xSmartContract
56. bin2chen
57. Deivitto
58. simon135
59. rbserver
60. Heuss
61. ReyAdmirado
62. gogo
63. mcwildy
64. sakman
65. pedr02b2
66. __141345__
67. pfapostol

68. neko_nyaa
69. ret2basic
70. Shishigami
71. halden
72. ch0bu
73. lucacez
74. c3phas
75. cryptostellar5
76. Shinchan (Sm4rty, prasantgupta52, and Rohan16)
77. adriro
78. Pheonix
79. ajtra
80. medikko

This contest was judged by Alex the Entreprenerd.

Final report assembled by liveactionllama.

## Summary

The C4 analysis yielded an aggregated total of 2 unique vulnerabilities. Of these vulnerabilities, 1 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 24 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 34 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 Blur Exchange contest repository, and is composed of 10 smart contracts written in the Solidity programming language and includes 801 lines of Solidity code.

## Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into 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

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website.

## High Risk Findings (1)

### [H-01] `StandardPolicyERC1155.sol` returns `amount == 1` instead of `amount == order.amount`

*Submitted by dipp, also found by 0x4non, 0x52, 0xc0ffEE, 0xRobocop, 8olidity, arcoun, aviggiano, bardamu, Ch_301, cryptonue, csanuragjain, d3e4, enckrish, exd0tpy, hansfriese, jayphbee, Jeiwan, joestakey, Junnon, KIntern_NA, ladboy233, Lambda, M4TZ1P, MiloTruck, minhquanym, minhtrng, nicobevi, Nyx, obront, PaludoX0, polymorphism, rokinot, romand, rotcivegaf, Ruhum, RustyRabbit, rvierdiiev, saian, serial-coder, Soosh, TomJ, Trust, trustindistrust, and zzykxx*

StandardPolicyERC1155.sol#L12-L36
BlurExchange.sol#L154-L161

The `canMatchMakerAsk` and `canMatchMakerBid` functions in `StandardPolicyERC1155.sol` will only return 1 as the amount instead of the order.amount value. This value is then used in the `_executeTokenTransfer` call during the execution flow and leads to only 1 ERC1155 token being sent. A buyer matching an ERC1155 order wih amount > 1 would expect to receive amount of tokens if they pay the order's price. The seller, who might also expect more than 1 tokens to be sent, would have set the order's price to be for the amount of tokens and not just for 1 token.

The buyer would lose overspent ETH/WETH to the seller without receiving all tokens as specified in the order.

## Proof of Concept

StandardPolicyERC1155.sol:canMatchMakerAsk

```
    function canMatchMakerAsk(Order calldata makerAsk, Order cal
        external
        pure
        override
        returns (
            bool,
            uint256,
            uint256,
            uint256,
            AssetType
        )
    {
        return (
            (makerAsk.side != takerBid.side) &&
            (makerAsk.paymentToken == takerBid.paymentToken) &&
            (makerAsk.collection == takerBid.collection) &&
            (makerAsk.tokenId == takerBid.tokenId) &&
            (makerAsk.matchingPolicy == takerBid.matchingPolicy)
            (makerAsk.price == takerBid.price),
            makerAsk.price,
            makerAsk.tokenId,
            1,
            AssetType.ERC1155
        );
    }
```

The code above shows that `canMatchMakerAsk` only returns 1 as the amount. `_executeTokenTransfer` will then call the executionDelegate's `transferERC1155` function with only amount 1, transferring only 1 token to the buyer.

Test code added to `execution.test.ts`:

```
it('Only 1 ERC1155 received for order with amount > 1', asyn
  await mockERC1155.mint(alice.address, tokenId, 10);
  sell = generateOrder(alice, {
    side: Side.Sell,
    tokenId,
    amount: 10,
    collection: mockERC1155.address,
    matchingPolicy: matchingPolicies.standardPolicyERC1155.a
  });
  buy = generateOrder(bob, {
    side: Side.Buy,
    tokenId,
    amount: 10,
    collection: mockERC1155.address,
    matchingPolicy: matchingPolicies.standardPolicyERC1155.a
  });
  sellInput = await sell.pack();
  buyInput = await buy.pack();

  await waitForTx(exchange.execute(sellInput, buyInput));

  // Buyer only receives 1 token
  expect(await mockERC1155.balanceOf(bob.address, tokenId)).
  await checkBalances(
    aliceBalance,
    aliceBalanceWeth.add(priceMinusFee),
    bobBalance,
    bobBalanceWeth.sub(price),
    feeRecipientBalance,
    feeRecipientBalanceWeth.add(fee),
  );
});
```

The test code above shows a sell order for an ERC1155 token with amount = 10 and a matching buy order. The `execute` function in `BlurExchange.sol` is called and the orders are matched but the buyer (bob) only receives 1 token instead of 10 despite paying the full price.

### Recommended Mitigation Steps

Policies used for ERC1155 tokens should return and consider the amount of tokens set for the order.

**blur-io-toad (Blur) acknowledged and commented:**

> This was an oversight on my part for not putting this contract as out-of-scope. Our marketplace does not handle ERC1155 yet and so we haven't concluded what the matching critieria for those orders will be. This contract was mainly created to test ERC1155 transfers through the rest of the exchange, but shouldn't be deployed initially. When we are prepared to handle ERC1155 orders we will have to develop a new matching policy that determines the amount from the order parameters. Acknowledging that it's incorrect, but won't be making any changes as the contract won't be deployed.

**Alex the Entreprenerd (Judge) commented:**

> The sponsor acknowledges the finding, and the report to be technically correct. However the sponsor claims they won't be using the code in production.

> Because the code is technically incorrect and was in scope during the contest am going to assign High Severity.
> However, I do understand that the contract will not be deployed.

**Alex the Entreprenerd (Judge) commented:**

> Despite the fact that some reports mention a slightly different risk than this one (mismatching amounts), given https://github.com/code-423n4/org/issues/8 and given the consideration that these are substantially the same issue (the policy has a hardcoded amount), am going to group them under the same issue.

> Because this report shows both sides of the issue, is well-written and has a coded Poc, am choosing to make it the selected report.

## Medium Risk Findings (1)

### [M-01] Contract Owner Possesses Too Many Privileges

*Submitted by Soosh, also found by 0x1f8b, 0x4non, 0xc0ffEE, arcoun, cccz, d3e4, minhtrng, RaymondFam, rotcivegaf, Ruhum, RustyRabbit, saian, sakshamguruji, and Trust*

ExecutionDelegate.sol#L119

To use the protocol (buy/sell NFTs), users must approve the `ExecutionDelegate` to handle transfers for their `ERC721`, `ERC1155`, or `ERC20` tokens.

The safety mechanisms mentioned by the protocol do not protect users at all if the project's owner decides to rugpull.

From the contest page, Safety Features:

- The calling contract must be approved on the `ExecutionDelegate`
- Users have the ability to revoke approval from the `ExecutionDelegate` without having to individually calling every token contract.

#### Proof of Concept

```
    function transferERC20(address token, address from, address to,
            approvedContract
            external
            returns (bool)
        {
            require(revokedApproval[from] == false, "User has revoke
            return IERC20(token).transferFrom(from, to, amount);
        }
```

The owner can set `approvedContract` to any address at any time with `approveContract(address _contract)`, and `revokeApproval()` can be frontrun. As a result, all user funds approved to the `ExecutionDelegate` contract can be lost via rugpull.

## Justification

While rug-pulling may not be the project's intention, I find that this is still an inherently dangerous design.

I am unsure about the validity of centralization risk findings on C4, but I argue this is a valid High risk issue as:

- It is too easy to steal all of user funds as a project owner. A single Bored Ape NFT traded on the exchange would mean roughly `$200,000` can be stolen based on current floor price (75.6 ETH as of writing, Source: [https://nftpricefloor.com/bored-ape-yacht-club](https://nftpricefloor.com/bored-ape-yacht-club)). `$200k` because 75.6ETH for NFT seller and at least 75.6ETH approved by buyer.
- web3 security should not be based on "trust".
- Assuming the project owner is not malicious and will never rug-pull:
    - 1 successful phishing attack (private key compromise) against the project's owner is all it takes to wipe the protocol out.
    - The protocol is still affected as user's will not want to trade on a platfrom knowing such an attack is possible.

## Recommended Mitigation Steps

This is due to an insecure design of the protocol. So as far as recommendations go, the team should reconsider the protocol's design.

I do not think `ExecutionDelegate` should be used. It would be better if `BlurExchange.sol` is approved by users instead. The exchange should require that the buyer has received their NFT and the seller has received their ETH/WETH or revert.

**Alex the Entreprenerd (Judge) decreased severity and commented:**

> Refactoring.

**Alex the Entreprenerd (Judge) increased severity to Medium and commented:**

> Per discussion in https://github.com/code-423n4/org/issues, as well as discussion at End of Contest Triage.

> Am changing the judging on these issues, as these reports have shown a risk to end-users and have historically rated Admin Privilege as a Medium Severity.

> Am making this the primary as it clearly shows the risk for end users.

## Low Risk and Non-Critical Issues

For this contest, 24 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by **0x4non** received the top score from the judge.

*The following wardens also submitted reports: 0x1f8b, 0xNazgul, 0xSmartContract, arcoun, bin2chen, zzykxx, brgltd, csanuragjain, d3e4, Deivitto, Trust, enckrish, exd0tpy, lllllll, ladboy233, Lambda, simon135, nicobevi, RaymondFam, rbserver, Rolezn, rotcivegaf, and RustyRabbit.*

### Remove `pragma abicoder v2`

You are using solidity v0.8 since that version and above the ABIEncoderV2 is not experimental anymore - it is actually enabled by default by the compiler.
Remove lines;
BlurExchange.sol#L3
ExecutionDelegate.sol#L3
interfaces/IBlurExchange.sol#L3
test/TestBlurExchange.sol#L3

### Use latest open zeppelin contracts

Your current version of `@openzeppelin/contracts` is 4.4.1 and latest version is 4.7.3
Your current version of `@openzeppelin/contracts-upgradeable` is ^4.6.0 and latest version is ^4.7.3

### Use named imports instead of plain import ‘file.sol’

You use regular imports on lines:
BlurExchange.sol/#L5-L16
ExecutionDelegate.sol#L5-L8
lib/ERC1967Proxy.sol#L5-L6

Instead of this, use named imports as you do on for example;
PolicyManager.sol#L4-L7
lib/EIP712.sol#L4
BlurExchange.sol#L17-L24

## Add gap to reserve space on upgradeble contracts to add new variables

On import lib/ReentrancyGuarded.sol and lib/EIP712.sol add this lines;

```
/**
 * @dev This empty reserved space is put in place to allow f
 * variables without shifting down storage in the inheritanc
 * See https://docs.openzeppelin.com/contracts/4.x/upgradeak
 */
uint256[50] private __gap;
```

In case you need to add variables for an upgrade you will have reserved space.

## On `BlurExchange.sol`

### Remove unused import `ERC20.sol`

On line contracts/BlurExchange.sol#L8

### Be explicit declaring types

On BlurExchange.sol/#L96 and BlurExchange.sol/#L101 instead of `uint` use `uint256`

### Avoid passing as reference the `chainid`

On BlurExchange.sol/#L96.
Instead of passing `chainid` as reference you could get current `chainid` using `block.chainid`
Please see;
https://docs.soliditylang.org/en/v0.8.0/units-and-global-variables.html#block-and-transaction-properties

### Missing `address(0)` checks for `_executionDelegate`, `_policyManager` and `_oracle`

By mistake any of these could be `address(0)` the could be chaged later by and admin, however is a good practice to check for `address(0)`
BlurExchange.sol/#L98-L100
Recommendation, add address(0) check;

```
    require(address(_VARIABLE) != address(0), "Address cannot be zer
```

### Missing `address(0)` for `_weth`

`_weth` variable on contracts/BlurExchange.sol/#L97 could be set as `address(0)` as mistake and there is no way to change it.
Recommendation, add address(0) check;

```
    require(address(_weth) != address(0), "Address cannot be zero");
```

### If `blockRange` is `0`, `_validateSignatures` will always fail updating oracle

Inspecting the functions that set `blockRange` on BlurExchange.sol#L117 and BlurExchange.sol#L246 it seems that `blockRange` can be `0`.

But if you set `blockRange` to `0` the condition that checks oracle authoriozation in line 318 will always fail;
`require(block.number - order.blockNumber < blockRange, "Signed block number out of range");`

Recommendation: add `<=` to the require, or create a minimal blockRange required.

## Revert if `ecrecover` is address(0)

On BlurExchange.sol#L408 add a revert that triggers if the response is address(0), this means that signature its not valid.

Example, by definition `oracle` could be initialized with address(0), then you will always can pass this line (oracle validation);
BlurExchange.sol#L392

```
return _recover(oracleHash, v, r, s) == oracle;
```

And also you could end up stealing, because is used on `_validateSignatures` BlurExchange.sol#L320 and this is also used on the main `execute` function that transfers nfts and tokens BlurExchange.sol#L128

1. avoid oracle to be address(0)
2. revert if ecrecover is address(0)
3. use openzepellin implementatio to reduce audit lines and headaches

## Avoid using low call function `ecrecover`

Use OZ library ECDSA that its battle tested to avoid classic errors.
contracts/utils/cryptography/ECDSA.sol
https://docs.openzeppelin.com/contracts/4.x/api/utils#ECDSA

## Empty revert message

On BlurExchange.sol/#L134, BlurExchange.sol#L183 and BlurExchange.sol#L452 there is no revert message. It is very important to add a message, so the user has enough information to know the reason of failure.

## Possible DOS out of gas on `_transferFees` function loop

This loop could drain all user gas and revert;
https://github.com/code-423n4/2022-10-blur/blob/main/contracts/BlurExchange.sol#L476-L479

## No validation on `fees`

Fees can have any desiree amount. Recommendation create a threashold to avoid excessive fees.
[BlurExchange.sol/#L477]()

### Use Checks-effects-interactions pattern on `execute`

Just move the `cancelledOrFilled` setting to stick to the Checks-effects-interactions pattern.

```
--- a/contracts/BlurExchange.sol
+++ b/contracts/BlurExchange.sol
@@ -144,6 +144,10 @@ contract BlurExchange is IBlurExchange, Ree

         (uint256 price, uint256 tokenId, uint256 amount, AssetT

+        /* Mark orders as filled. */
+        cancelledOrFilled[sellHash] = true;
+        cancelledOrFilled[buyHash] = true;
+
         _executeFundsTransfer(
             sell.order.trader,
             buy.order.trader,
@@ -160,10 +164,6 @@ contract BlurExchange is IBlurExchange, Ree
             assetType
         );

-        /* Mark orders as filled. */
-        cancelledOrFilled[sellHash] = true;
-        cancelledOrFilled[buyHash] = true;
-
         emit OrdersMatched(
             sell.order.listingTime <= buy.order.listingTime ? s
             sell.order.listingTime > buy.order.listingTime ? se
```

### Use OZ MerkleTree implementation instead of creating a new one

Instead of your own merkle tree lib, [BlurExchange.sol#L12]() Use openzeppelin implementation;
[https://docs.openzeppelin.com/contracts/4.x/api/utils#MerkleProof](https://docs.openzeppelin.com/contracts/4.x/api/utils#MerkleProof)

### Use OZ eip712 instead of creating your onw implementation

Use openzepellin implementation, and save a lot of possible bugs by writing your own implementation;
Code is in;
https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/utils/cryptography/EIP712Upgradeable.sol

Just;

```
    import "@openzeppelin/contracts-upgradeable/utils/cryptography/E
```

https://docs.openzeppelin.com/contracts/4.x/api/utils#EIP712
(docs could be old, it's not on draft anymore)

## Contract EIP712 should be marked as abstract

Contract EIP712 should be abstract

Alex the Entreprenerd (Judge) commented:

> **Remove pragma abicoder v2**
> `Non-Critical`

> **Use latest open zeppelin contracts**
> `Non-Critical`

> **Use named imports instead of plain** `import file.sol`
> `Refactoring`

> **Add gap to reserve space on upgradeble contracts to add new variables**
> `Low`

> **Remove unused import ERC20.sol**
> `Non-Critical`

> **Be explicit declaring types**
> Valid. `Non-Critical`

> **Avoid passing as reference the chainid**
>
> `Low`

> **Missing `address(0)` checks for `_executionDelegate`, `_policyManager` and `_oracle`**
>
> `Low`

> **If blockRange is 0, `_validateSignatures` will always fail updating oracle**
>
> `Non-Critical`

> **Revert if ecrecover is address(0)**
>
> `Refactoring`

> **Avoid using low call function ecrecover**
>
> `Refactoring`

> **Empty revert message**
>
> `Non-Critical`

> **Possible DOS out of gas on `_transferFees` function loop**
>
> `Low`

> **No validation on fees**
>
> `Low`

> **Use Checks-effects-interactions pattern on execute**
>
> `Refactoring`

> **Contract EIP712 should be marked as abstract**
>
> `Refactoring`

> Pretty good thorough report.

> 5 Low, 5 Refactoring, 6 Non-Critical

**Alex the Entreprenerd (Judge) commented:**

> After adding the downgraded findings this report won by quite a strong margin (10+ points, 20%)

> Well played!

*Note: see this warden's original submission for full notes from the judge.*

## Gas Optimizations

For this contest, 34 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:* 0x1f8b, RaymondFam, Heuss, Ruhum, rvierdiiev, Lambda, ReyAdmirado, gogo, mcwildy, sakman, pedr02b2, enckrish, __141345__, pfapostol, saian, 0xSmartContract, neko_nyaa, ret2basic, sakshamguruji, Shishigami, halden, ch0bu, Aymen0909, lucacez, c3phas, 0xNazgul, cryptostellar5, Shinchan, adriro, Pheonix, ajtra, d3e4, *and* medikko.

## Summary

### Gas Optimizations

| | Issue | Instances | Total Gas Saved |
|---|---|---|---|
| [G-01] | State variables that never change should be declared `immutable` or `constant` | 1 | 2097 |
| [G-02] | Multiple `address`/ID mappings can be combined into a single `mapping` of an `address`/ID to a `struct`, where appropriate | 1 | 8400 |
| [G-03] | Structs can be packed into fewer storage slots | 1 | - |
| [G-04] | Using `calldata` instead of `memory` for read-only arguments in `external` functions saves gas | 1 | 120 |
| [G-05] | State variables should be cached in stack variables rather than re-reading them from storage | 1 | 97 |
| [G-06] | The result of function calls should be cached rather than re-calling the function | 1 | 17 |

| | Issue | Instances | Total Gas Saved |
|---|---|---|---|
| [G-07] | `internal` functions only called once can be inlined to save gas | 10 | 200 |
| [G-08] | Add `unchecked {}` for subtractions where the operands cannot underflow because of a previous `require()` or `if` -statement | 1 | 85 |
| [G-09] | `<array>.length` should not be looked up in every loop of a `for` -loop | 4 | 12 |
| [G-10] | `++i` / `i++` should be `unchecked{++i}` / `unchecked{i++}` when it is not possible for them to overflow, as is the case when used in `for` - and `while` -loops | 5 | 300 |
| [G-11] | `require()` / `revert()` strings longer than 32 bytes cost extra gas | 2 | - |
| [G-12] | Optimize names to save gas | 3 | 66 |
| [G-13] | Using `bool` s for storage incurs overhead | 4 | 51300 |
| [G-14] | `++i` costs less gas than `i++` , especially when it's used in `for` -loops ( `--i` / `i--` too) | 5 | 25 |
| [G-15] | Using `private` rather than `public` for constants, saves gas | 7 | - |
| [G-16] | Don't compare boolean expressions to boolean literals | 5 | 45 |
| [G-17] | Use custom errors rather than `revert()` / `require()` strings to save gas | 23 | - |
| [G-18] | Functions guaranteed to revert when called by normal users can be marked `payable` | 11 | 231 |

Total: 86 instances over 18 issues with **62,995 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-01] State variables that never change should be declared `immutable` or `constant`

Avoids a Gsset (**20000 gas**) in the constructor, and replaces the first access in each transaction (Gcoldsload - **2100 gas**) and each access thereafter (Gwarmacces - **100 gas**) with a `PUSH32` (**3 gas**).

While it's not possible to use `immutable` because the contract is UUPS, the variable can be declared as a hard-coded `constant` and get the same gas savings.

*There is 1 instance of this issue:*

```
File: /contracts/BlurExchange.sol

509           } else if (paymentToken == weth) {
510               /* Transfer funds in WETH. */
511:              executionDelegate.transferERC20(weth, from, to,
```

[https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/BlurExchange.sol#L509-L511](https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/BlurExchange.sol#L509-L511)

## [G-02] 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.

All functions that check `contracts` also check `revokedApproval`, which means if the modifier is changed to check both, both the ~42 gas and the ~2100 gas savings get applied. There are four instances of the `approvedContract()` modifier, so making such a change saves approximately **8,400 gas**.

*There is 1 instance of this issue:*

```
File: contracts/ExecutionDelegate.sol

18        mapping(address => bool) public contracts;
19:       mapping(address => bool) public revokedApproval;
```

https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/ExecutionDelegate.sol#L18-L19

## [G-03] Structs can be packed into fewer storage slots

Each slot saved can avoid an extra Gsset (**20000 gas**) for the first setting of the struct. Subsequent reads as well as writes have smaller gas savings.

*There is 1 instance of this issue:*

```
File: contracts/lib/OrderStructs.sol

/// @audit Variable ordering with 6 slots instead of the current
///           user-defined(32):order, bytes32(32):r, bytes32(32)
30    struct Input {
31        Order order;
32        uint8 v;
33        bytes32 r;
34        bytes32 s;
35        bytes extraSignature;
36        SignatureVersion signatureVersion;
37        uint256 blockNumber;
38:   }
```

https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/lib/OrderStructs.sol#L30-L38

## [G-04] 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 is 1 instance of this issue:*

```
File: contracts/lib/MerkleVerifier.sol

/// @audit proof
17        function _verifyProof(
18            bytes32 leaf,
19            bytes32 root,
20:           bytes32[] memory proof
```

[https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/lib/MerkleVerifier.sol#L17-L20](https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/lib/MerkleVerifier.sol#L17-L20)

## [G-05] 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 is 1 instance of this issue:*

```
File: contracts/BlurExchange.sol

/// @audit weth on line 509
511:                executionDelegate.transferERC20(weth, from, to
```

[https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/BlurExchange.sol#L511](https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/BlurExchange.sol#L511)

## [G-06] The result of function calls should be cached rather than re-calling the function

The instances below point to the second+ call of the function within a single function

*There is 1 instance of this issue:*

```
File: contracts/PolicyManager.sol

/// @audit _whitelistedPolicies.length() on line 71
72:                length = _whitelistedPolicies.length() - curso
```

[https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/PolicyManager.sol#L72](https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/PolicyManager.sol#L72)

## [G-07] `internal` functions only called once can be inlined to save gas

Not inlining costs **20 to 40 gas** because of two extra `JUMP` instructions and additional stack operations needed for function calls.

*There are 10 instances of this issue:*

```
File: contracts/BlurExchange.sol

278        function _canSettleOrder(uint256 listingTime, uint256
279            view
280            internal
281:           returns (bool)

344        function _validateUserAuthorization(
345            bytes32 orderHash,
346            address trader,
347            uint8 v,
348            bytes32 r,
349            bytes32 s,
350            SignatureVersion signatureVersion,
351            bytes calldata extraSignature
352:       ) internal view returns (bool) {

375        function _validateOracleAuthorization(
376            bytes32 orderHash,
377            SignatureVersion signatureVersion,
378            bytes calldata extraSignature,
379            uint256 blockNumber
380:       ) internal view returns (bool) {

416        function _canMatchOrders(Order calldata sell, Order ca
417            internal
418            view
419:           returns (uint256 price, uint256 tokenId, uint256 a

444        function _executeFundsTransfer(
445            address seller,
446            address buyer,
447            address paymentToken,
448            Fee[] calldata fees,
449:           uint256 price

469        function _transferFees(
470            Fee[] calldata fees,
471            address paymentToken,
472            address from,
473            uint256 price
474:       ) internal returns (uint256) {
```

```
525        function _executeTokenTransfer(
526            address collection,
527            address from,
528            address to,
529            uint256 tokenId,
530            uint256 amount,
531:           AssetType assetType

548        function _exists(address what)
549            internal
550            view
551:           returns (bool)
```

https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/BlurExchange.sol#L278-L281

```
File: contracts/lib/EIP712.sol

55         function _hashFee(Fee calldata fee)
56             internal
57             pure
58:            returns (bytes32)

69         function _packFees(Fee[] calldata fees)
70             internal
71             pure
72:            returns (bytes32)
```

https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/lib/EIP712.sol#L55-L58

## [G-08] Add `unchecked {}` for subtractions where the operands cannot underflow because of a previous `require()` or `if`-statement

`require(a <= b); x = b - a` => `require(a <= b); unchecked { x = b - a }`

*There is 1 instance of this issue:*

```
File: contracts/BlurExchange.sol

/// @audit require() on line 482
485:            uint256 receiveAmount = price - totalFee;
```

[https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/BlurExchange.sol#L485](https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/BlurExchange.sol#L485)

## [G-09] `<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 4 instances of this issue:*

```
File: contracts/BlurExchange.sol

199:            for (uint8 i = 0; i < orders.length; i++) {

476:            for (uint8 i = 0; i < fees.length; i++) {
```

[https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/BlurExchange.sol#L199](https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/BlurExchange.sol#L199)

```
File: contracts/lib/EIP712.sol
```

```
77:            for (uint256 i = 0; i < fees.length; i++) {
```

https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/lib/EIP712.sol#L77

```
File: contracts/lib/MerkleVerifier.sol

38:            for (uint256 i = 0; i < proof.length; i++) {
```

https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/lib/MerkleVerifier.sol#L38

## [G-10] `++i` / `i++` should be `unchecked{++i}` / `unchecked{i++}` when it is not possible for them to overflow, as is the case when used in `for` - and `while` -loops

The `unchecked` keyword is new in solidity version 0.8.0, so this only applies to that version or higher, which these instances are. This saves **30-40 gas** per loop.

*There are 5 instances of this issue:*

```
File: contracts/BlurExchange.sol

199:            for (uint8 i = 0; i < orders.length; i++) {

476:            for (uint8 i = 0; i < fees.length; i++) {
```

https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/BlurExchange.sol#L199

```
File: contracts/lib/EIP712.sol

77:            for (uint256 i = 0; i < fees.length; i++) {
```

[https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/lib/EIP712.sol#L77](https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/lib/EIP712.sol#L77)

```
File: contracts/lib/MerkleVerifier.sol

38:            for (uint256 i = 0; i < proof.length; i++) {
```

[https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/lib/MerkleVerifier.sol#L38](https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/lib/MerkleVerifier.sol#L38)

```
File: contracts/PolicyManager.sol

77:            for (uint256 i = 0; i < length; i++) {
```

[https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/PolicyManager.sol#L77](https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/PolicyManager.sol#L77)

## [G-11] `require()` / `revert()` strings longer than 32 bytes cost extra gas

Each extra memory word of bytes past the original 32 [incurs an MSTORE](https://gist.github.com/hrkrshnn/ee8fabd532058307229d65dcd5836ddc#consider-having-short-revert-strings) which costs **3 gas**

*There are 2 instances of this issue:*

```
File: contracts/BlurExchange.sol

482:            require(totalFee <= price, "Total amount of fees a
```

https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/BlurExchange.sol#L482

```
File: contracts/ExecutionDelegate.sol

22:         require(contracts[msg.sender], "Contract is not ap
```

https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/ExecutionDelegate.sol#L22

## [G-12] 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 3 instances of this issue:*

```
File: contracts/BlurExchange.sol

/// @audit open(), close(), initialize(), execute(), cancelOrder
30:   contract BlurExchange is IBlurExchange, ReentrancyGuarded,
```

https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/BlurExchange.sol#L30

```
File: contracts/ExecutionDelegate.sol

/// @audit approveContract(), denyContract(), revokeApproval(),
16:   contract ExecutionDelegate is IExecutionDelegate, Ownable
```

[https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/ExecutionDelegate.sol#L16](https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/ExecutionDelegate.sol#L16)

```
File: contracts/lib/MerkleVerifier.sol

/// @audit _verifyProof(), _computeRoot()
8:    library MerkleVerifier {
```

[https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/lib/MerkleVerifier.sol#L8](https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/lib/MerkleVerifier.sol#L8)

## [G-13] 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 tc
    // slot's contents, replace the bits taken up by the boolear
    // back. This is the compiler's defense against contract up
    // pointer aliasing, and it cannot be disabled.
```

[https://github.com/OpenZeppelin/openzeppelin-contracts/blob/58f635312aa21f947cae5f8578638a85aa2519f5/contracts/security/ReentrancyGuard.sol#L23-L27](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](https://gist.github.com/IllIllI000/1b70014db712f8572a72378321250058)) for the extra SLOAD, and to avoid Gsset (**20000 gas**) when changing from `false` to `true`, after having been `true` in the past.

*There are 4 instances of this issue:*

```
File: contracts/BlurExchange.sol

/// @audit excluding this one from the gas count since it's not
```

```
71:        mapping(bytes32 => bool) public cancelledOrFilled;
```

[https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/BlurExchange.sol#L71](https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/BlurExchange.sol#L71)

```
File: contracts/ExecutionDelegate.sol

18:        mapping(address => bool) public contracts;

19:        mapping(address => bool) public revokedApproval;
```

[https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/ExecutionDelegate.sol#L18](https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/ExecutionDelegate.sol#L18)

```
File: contracts/lib/ReentrancyGuarded.sol

10:        bool reentrancyLock = false;
```

[https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/lib/ReentrancyGuarded.sol#L10](https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/lib/ReentrancyGuarded.sol#L10)

## [G-14] `++i` costs less gas than `i++`, especially when it’s used in `for`-loops (`--i`/`i--` too)

Saves **5 gas per loop**.

*There are 5 instances of this issue:*

```
File: contracts/BlurExchange.sol

199:            for (uint8 i = 0; i < orders.length; i++) {

476:            for (uint8 i = 0; i < fees.length; i++) {
```

[https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/BlurExchange.sol#L199](https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/BlurExchange.sol#L199)

```
File: contracts/lib/EIP712.sol

77:            for (uint256 i = 0; i < fees.length; i++) {
```

[https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/lib/EIP712.sol#L77](https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/lib/EIP712.sol#L77)

```
File: contracts/lib/MerkleVerifier.sol

38:            for (uint256 i = 0; i < proof.length; i++) {
```

[https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/lib/MerkleVerifier.sol#L38](https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/lib/MerkleVerifier.sol#L38)

```
File: contracts/PolicyManager.sol

77:            for (uint256 i = 0; i < length; i++) {
```

[https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/PolicyManager.sol#L77](https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/PolicyManager.sol#L77)

## [G-15] Using `private` rather than `public` for constants, saves gas

If needed, the values can be read from the verified contract source code, or if there are multiple values there can be a single getter function that [returns a tuple]() of the values of all currently-public constants. Saves **3406-3606 gas** in deployment gas due to the compiler not having to create non-payable getter functions for deployment calldata, not having to store the bytes of the value outside of where it's used, and not adding another entry to the method ID table.

*There are 7 instances of this issue:*

```
File: contracts/BlurExchange.sol

57:        string public constant name = "Blur Exchange";

58:        string public constant version = "1.0";

59:        uint256 public constant INVERSE_BASIS_POINT = 10000;
```

[https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/BlurExchange.sol#L57](https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/BlurExchange.sol#L57)

```
File: contracts/lib/EIP712.sol

20         bytes32 constant public FEE_TYPEHASH = keccak256(
21             "Fee(uint16 rate,address recipient)"
22:        );

23         bytes32 constant public ORDER_TYPEHASH = keccak256(
24             "Order(address trader,uint8 side,address matching
25:        );

26         bytes32 constant public ORACLE_ORDER_TYPEHASH = keccak
27             "OracleOrder(Order order,uint256 blockNumber)Fee(
28:        );

29         bytes32 constant public ROOT_TYPEHASH = keccak256(
30             "Root(bytes32 root)"
31:        );
```

[https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/lib/EIP712.sol#L20-L22](https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/lib/EIP712.sol#L20-L22)

## [G-16] Don’t compare boolean expressions to boolean literals

`if (<x> == true)` => `if (<x>)`, `if (<x> == false)` => `if (!<x>)`

*There are 5 instances of this issue:*

```
File: contracts/BlurExchange.sol

267:                (cancelledOrFilled[orderHash] == false) &&
```

[https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/BlurExchange.sol#L267](https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/BlurExchange.sol#L267)

```
File: contracts/ExecutionDelegate.sol

77:           require(revokedApproval[from] == false, "User has

92:           require(revokedApproval[from] == false, "User has

108:          require(revokedApproval[from] == false, "User has

124:          require(revokedApproval[from] == false, "User has
```

[https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/ExecutionDelegate.sol#L77](https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/ExecutionDelegate.sol#L77)

## [G-17] Use custom errors rather than `revert()` / `require()` strings to save gas

Custom errors are available from solidity version 0.8.4. Custom errors save ~50 gas each time they’re hit by avoiding having to allocate and store the revert string. Not defining the strings also save deployment gas.

*There are 23 instances of this issue:*

```
File: contracts/BlurExchange.sol

36:             require(isOpen == 1, "Closed");

139:            require(_validateOrderParameters(sell.order, sellH

140:            require(_validateOrderParameters(buy.order, buyHas

142:            require(_validateSignatures(sell, sellHash), "Sell

143:            require(_validateSignatures(buy, buyHash), "Buy fa

219:            require(address(_executionDelegate) != address(0),

228:            require(address(_policyManager) != address(0), "Ac

237:            require(_oracle != address(0), "Address cannot be

318:                require(block.number - order.blockNumber < blo

407:            require(v == 27 || v == 28, "Invalid v parameter")

424:                require(policyManager.isPolicyWhitelisted(sell

428:                require(policyManager.isPolicyWhitelisted(buy.

431:            require(canMatch, "Orders cannot be matched");

482:            require(totalFee <= price, "Total amount of fees a

534:            require(_exists(collection), "Collection does not
```

https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/BlurExchange.sol#L36

```
File: contracts/ExecutionDelegate.sol

22:             require(contracts[msg.sender], "Contract is not ap

77:             require(revokedApproval[from] == false, "User has

92:             require(revokedApproval[from] == false, "User has
108:            require(revokedApproval[from] == false, "User has

124:            require(revokedApproval[from] == false, "User has
```

https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/ExecutionDelegate.sol#L22

```
File: contracts/lib/ReentrancyGuarded.sol

14:            require(!reentrancyLock, "Reentrancy detected");
```

https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/lib/ReentrancyGuarded.sol#L14

```
File: contracts/PolicyManager.sol

26:            require(!_whitelistedPolicies.contains(policy), "A

37:            require(_whitelistedPolicies.contains(policy), "Nc
```

https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/PolicyManager.sol#L26

## [G-18] 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), `REVER`

`T` (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 11 instances of this issue:*

```
File: contracts/BlurExchange.sol

43:        function open() external onlyOwner {

47:        function close() external onlyOwner {

53:        function _authorizeUpgrade(address) internal override

215        function setExecutionDelegate(IExecutionDelegate _exec
216            external
217:           onlyOwner

224        function setPolicyManager(IPolicyManager _policyManage
225            external
226:           onlyOwner

233        function setOracle(address _oracle)
234            external
235:           onlyOwner

242        function setBlockRange(uint256 _blockRange)
243            external
244:           onlyOwner
```

[https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/BlurExchange.sol#L43](https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/BlurExchange.sol#L43)

```
File: contracts/ExecutionDelegate.sol

36:        function approveContract(address _contract) onlyOwner

45:        function denyContract(address _contract) onlyOwner ext
```

https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/ExecutionDelegate.sol#L36

```
File: contracts/PolicyManager.sol

25:        function addPolicy(address policy) external override c

36:        function removePolicy(address policy) external overric
```

https://github.com/code-423n4/2022-10-blur/blob/2fdaa6e13b544c8c11d1c022a575f16c3a72e3bf/contracts/PolicyManager.sol#L25

**Alex the Entreprenerd (Judge) commented:**

> 2.1k from weth
> 500 memory
> 5k from reentancy

> Rest is 150

> Disagree with G-02 as they are meant to be separate.

> 7750

**Alex the Entreprenerd (Judge) commented:**

> Ultimately best submission as it contained both high leverage savings + some minor savings that did add up to win, wp.

## 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