Delegate
Findings & Analysis Report

2023-11-15

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 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 outlined in this document, C4 conducted an analysis of the Delegate smart contract system written in Solidity. The audit took place between September 5—September 11 2023.

Wardens

17 Wardens contributed reports to the Delegate:

1. ladboy233
2. d4r3d3v1l
3. DadeKuma
4. pfapostol
5. Sathish9098
6. Baki (Viraz and supernova)
7. p0wd3r
8. m4ttm
9. BanditxOx
10. gkrastenov
11. Fulum
12. sces60107
13. Brenzee
14. kodyvim
15. lsaudit
16. lodelux

This audit 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, 0 received a risk rating in the category of HIGH severity and 2 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 10 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 2 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 Delegate repository, and is composed of 12 smart contracts written in the Solidity programming language and includes 1,824 lines of Solidity code.

In addition to the known issues identified by the project team, a Code4rena bot race was conducted at the start of the audit. The winning bot, **Hound** from warden DadeKuma, generated the Automated Findings report and all findings therein were classified as out of scope.

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 (2)

[M-01] No way to revoke Approval in `DelegateToken.approve` leads to unauthorized calling of `DelegateToken.transferFrom`

Submitted by d4r3d3v1l

https://github.com/code-423n4/2023-09-delegate/blob/main/src/DelegateToken.sol#L134
https://github.com/code-423n4/2023-09-delegate/blob/main/src/DelegateToken.sol#L168
https://github.com/code-423n4/2023-09-delegate/blob/main/src/libraries/DelegateTokenStorageHelpers.sol#L149

There is no way to revoke the approval which given via `DelegateToken.approve(address,delegateTokenId)`. They can able call the `DelegateToken.transferFrom` even the tokenHolder revoke the permission using the `DelegateToken.setApprovalForAll`.

If the spender address is approved by the PT token, we can call the `DelegateToken.withdraw`.

Proof of Concept

Alice is the token Holder.

Alice approves Bob via `DelegateToken.setApprovalForAll(Bob,true)`.

Bob approves himself using `DelegateToken.approve(Bob,delegateTokenId)`

Alice revokes the Bob approval by calling `DelegateToken.setApprovalForAll(Bob,false);`

Now Bob can still calls the `DelegateToken.transferFrom(Alice,to,delegateTokenId)` which is subjected to call only by approved address.

The transfer will be successful.

Code details

https://github.com/code-423n4/2023-09-delegate/blob/main/src/libraries/DelegateTokenStorageHelpers.sol#L143

```
    function revertNotApprovedOrOperator(
```

```
        mapping(address account => mapping(address operator => b
        mapping(uint256 delegateTokenId => uint256[3] info) stor
        address account,
        uint256 delegateTokenId
    ) internal view {
        if (msg.sender == account || accountOperator[account][ms
        revert Errors.NotApproved(msg.sender, delegateTokenId);
    }
```

Even after revoking the approval for operator using `setApprovalAll` this `msg.sender == readApproved(delegateTokenInfo, delegateTokenId)` will be true and able to call `transferFrom` function.

Test function

```
function testFuzzingTransfer721(
        address from,
        address to,
        uint256 underlyingTokenId,
        bool expiryTypeRelative,
        uint256 time
    ) public {
        vm.assume(from != address(0));
        vm.assume(from != address(dt));

        (
            ,
            /* ExpiryType */
            uint256 expiry, /* ExpiryValue */

        ) = prepareValidExpiry(expiryTypeRelative, time);
        mockERC721.mint(address(from), 33);

        vm.startPrank(from);
        mockERC721.setApprovalForAll(address(dt), true);

        vm.stopPrank();
        vm.prank(from);
        dt.setApprovalForAll(address(dt), true);
        vm.prank(from);
        uint256 delegateId1 = dt.create(
            DelegateTokenStructs.DelegateInfo(
                from,
```

```
                IDelegateRegistry.DelegationType.ERC721,
                from,
                0,
                address(mockERC721),
                33,
                "",
                expiry
            ),
            SALT + 1
        );

        vm.prank(address(dt));
        dt.approve(address(dt), delegateId1);

        vm.prank(from);

        dt.setApprovalForAll(address(dt), false);
        address tmp = dt.getApproved(delegateId1);
        console.log(tmp);
        vm.prank(address(dt));
        dt.transferFrom(from, address(0x320), delegateId1);
    }
```

Recommended Mitigation Steps

If token Holder revokes the approval for a operator using `DelegateToken.setApprovalForAll` , revoke the all the approvals(DelegateToken.approve) which is done by the operator.

Assessed type

Access Control

Oxfoobar (Delegate) confirmed and commented:

> Need to double-check, but looks plausible.

Alex the Entreprenerd (judge) commented:

> In contrast to ERC721, which only allows the owner to change `approve`
> ERC721.sol#L448-L454

> `DT.approve` allows the operator to set themselves as approved, which can technically be undone via a multicall but may not be performed under normal usage.

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> Leaning towards Medium Severity as the finding requires:
>
> - Alice `approvesForAll` Bob
> - Bob approves self
> - Alice revokes Bobs `approvalForAll`
> - Alice doesn't revoke the `_approve` that Bob gave to self
>
> Notice that any transfer would reset the approval as well.

0xfoobar (Delegate) commented:

> A note about how ERC721 works: there are two different types of approvals. `approve()` lets an account move a single tokenId, while `setApprovalForAll()` marks an address as an operator to move all tokenIds. We can note the difference in the core OpenZeppelin ERC721 base contract, two separate mappings: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC721/ERC721.sol#L32-L34

> So plausible that an operator could be permitted to add new tokenid-specific approvals, but see your point that OZ ERC721 doesn't allow this by default and so could lead to confusing states that are possible but not fun to get out of.

[M-02] `CreateOfferer.sol` should not enforce the nonce incremented sequentially, otherwise user can DOS the contract by skipping order

Submitted by ladboy233

According to https://github.com/ProjectOpenSea/seaport/blob/main/docs/SeaportDocumentation.md#contract-orders:

> “Seaport v1.2 introduced support for a new type of order: the contract order. In brief, a smart contract that implements the ContractOffererInterface (referred to as an “Seaport app contract” or ‘Seaport app’ in the docs and a ‘contract offerer’ in the code) can now provide a dynamically generated order (a contract order) in response to a buyer or seller’s contract order request.”

The CreateOfferer.sol aims to be comply with the interface ContractOffererInterface.

The life cycle of the contract life cycle here is here:

https://github.com/ProjectOpenSea/seaport/blob/main/docs/SeaportDocumentation.md#example-lifecycle-journey

First the function `_getGeneratedOrder` is called.

Then after the order execution, the function ratifyOrder is triggered for contract (CreateOfferer.sol) to do post order validation.

In the logic of ratifyOrder, the nonce is incremented by calling this line of code `Helpers.processNonce`.

```
    function ratifyOrder(SpentItem[] calldata offer, ReceivedItem[]
            external
            checkStage(Enums.Stage.ratify, Enums.Stage.generate)
            onlySeaport(msg.sender)
            returns (bytes4)
    {
            Helpers.processNonce(nonce, contractNonce);
            Helpers.verifyCreate(delegateToken, offer[0].identifier,
            return this.ratifyOrder.selector;
    }
```

This is calling this line of code.

```
    function processNonce(CreateOffererStructs.Nonce storage nonce,
            if (nonce.value != contractNonce) revert CreateOffererEr
            unchecked {
                    ++nonce.value;
            } // Infeasible this will overflow if starting point is
```

```
}
```

The CreateOffererStructs.Nonce data structure is just **nonce**.

```
library CreateOffererStructs {
    /// @notice Used to track the stage and lock status
    struct Stage {
        CreateOffererEnums.Stage flag;
        CreateOffererEnums.Lock lock;
    }

    /// @notice Used to keep track of the seaport contract nonce
    struct Nonce {
        uint256 value;
    }
```

But this is not how Seaport contract track contract nonce.

On Seaport contract, we are calling **_getGeneratedOrder**.

Which calls **_callGenerateOrder**.

```
{
        // Do a low-level call to get success status and any ret
        (bool success, bytes memory returnData) = _callGenerateO
                orderParameters,
                context,
                originalOfferItems,
                originalConsiderationItems
        );

        {
                // Increment contract nonce and use it to derive
                // Note: nonce will be incremented even for skip
                // even if generateOrder's return data doesn't m
                uint256 contractNonce = (
                        _contractNonces[orderParameters.offerer]
                );

                // Derive order hash from contract nonce and off
                orderHash = bytes32(
```

```
                        contractNonce ^
                                (uint256(uint160(orderParameters
                );
        }
```

As we can see:

```
    // Increment contract nonce and use it to derive order hash.
    // Note: nonce will be incremented even for skipped orders, and
    // even if generateOrder's return data doesn't meet constraints.
    uint256 contractNonce = (
            _contractNonces[orderParameters.offerer]++
    );
```

Nonce will be incremented even for skipped orders.

This is very important, suppose the low level call _callGenerateOrder return false, we are hitting the [else block](#).

```
     return _revertOrReturnEmpty(revertOnInvalid, orderHash);
```

This is calling `_revertOrReturnEmpty`.

```
    function _revertOrReturnEmpty(
            bool revertOnInvalid,
            bytes32 contractOrderHash
    )
            internal
            pure
            returns (
                    bytes32 orderHash,
                    uint256 numerator,
                    uint256 denominator,
                    OrderToExecute memory emptyOrder
            )
    {
            // If invalid input should not revert...
            if (!revertOnInvalid) {
                    // Return the contract order hash and zero value
```

```
                // and denominator.
                return (contractOrderHash, 0, 0, emptyOrder);
            }

            // Otherwise, revert.
            revert InvalidContractOrder(contractOrderHash);
        }
```

Clearly we can see that if the flag revertOnInvalid is set to false, then even the low level call return false, the nonce of the offerer is still incremented.

Where in the Seaport code does it set revertOnInvalid to false?

When the Seaport wants to combine multiple orders in [this line of code](#).

```
    function _fulfillAvailableAdvancedOrders(
        AdvancedOrder[] memory advancedOrders,
        CriteriaResolver[] memory criteriaResolvers,
        FulfillmentComponent[][] memory offerFulfillments,
        FulfillmentComponent[][] memory considerationFulfillment
        bytes32 fulfillerConduitKey,
        address recipient,
        uint256 maximumFulfilled
    ) internal returns (bool[] memory, /* availableOrders */ Exe
        // Validate orders, apply amounts, & determine if they u
        (bytes32[] memory orderHashes, bool containsNonOpen) = _
            advancedOrders,
            criteriaResolvers,
            false, // Signifies that invalid orders should NOT r
            maximumFulfilled,
            recipient
        );
```

We [call](#) `_validateOrderAndUpdateStatus`.

```
    // Validate it, update status, and determine fraction to fill.
    (bytes32 orderHash, uint256 numerator, uint256 denominator) =
    _validateOrderAndUpdateStatus(advancedOrder, revertOnInvalid);
```

Finally we call the logic _getGeneratedOrder(orderParameters, advancedOrder.extraData, revertOnInvalid) below with parameter revertOnInvalid false.

```
        // If the order is a contract order, return the generated orde
            if (orderParameters.orderType == OrderType.CONTRACT) {
                    // Ensure that the numerator and denominator are
                    assembly {
                            // (1 ^ nd =/= 0) => (nd =/= 1) => (n =/
                            // It's important that the values are 12
                            // multiplication is applied. Otherwise,
                            // above is not correct (mod 2^256).
                            invalidFraction := xor(mul(numerator, de
                    }

                    // Revert if the supplied numerator and denomina
                    if (invalidFraction) {
                            _revertBadFraction();
                    }

                    // Return the generated order based on the order
                    // provided extra data. If revertOnInvalid is tr
                    // will revert if the input is invalid.
                    return _getGeneratedOrder(orderParameters, advar
            }
```

Ok what does this mean?

Suppose the CreateOfferer.sol fulfills two orders and creates two delegate tokens.

The nonces start from 0 and then increment to 1 and then increment to 2.

A user crafts a contract with malformed minimumReceived and combines with another valid order to call OrderCombine.

As we can see above, when multiple order is passed in, the revertOnInvalid is set to false, so the contract order from CreateOfferer.sol is skipped, but the nonce is incremented.

Then the nonce tracked by CreateOfferer.sol internally is out of sync with the contract nonce in seaport contract forever.

Then the CreateOfferer.sol is not usable because if the ratifyOrder callback hit the contract, transaction revert in this check.

```
    if (nonce.value != contractNonce) revert CreateOffererErrors.Ir
```

Recommended Mitigation Steps

I would recommend do not validate the order execution in the ratifyOrder call back by using the contract nonce, instead, validate the order using orderHash.

```
    if (
    ContractOffererInterface(offerer).ratifyOrder(
    orderToExecute.spentItems,
    orderToExecute.receivedItems,
    advancedOrder.extraData,
    orderHashes,
    uint256(orderHash) ^ (uint256(uint160(offerer)) << 96)
    ) != ContractOffererInterface.ratifyOrder.selector
    )
```

Assessed type

DoS

Oxfoobar (Delegate) confirmed and commented:

> Need to double-check but looks plausible.

Alex the Entreprenerd (judge) commented:

> `_callGenerateOrder` is caught, and in case of multiple order will be incrementing nonce without reverting.

> ReferenceOrderValidator.sol#L286-L302

```
        function _callGenerateOrder(
            OrderParameters memory orderParameters,
            bytes memory context,
```

```
        SpentItem[] memory originalOfferItems,
        SpentItem[] memory originalConsiderationItems
    ) internal returns (bool success, bytes memory returnData) {
        return
            orderParameters.offerer.call(
                abi.encodeWithSelector(
                    ContractOffererInterface.generateOrder.selec
                    msg.sender,
                    originalOfferItems,
                    originalConsiderationItems,
                    context
                )
            );
    }
```

> The call to `generateOrder` will revert at `processSpentItems`.

> CreateOffererLib.sol#L351-L352

```
        if (!(minimumReceived.length == 1 && maximumSpent.length
```

> This will cause the nonce to be increased in Seaport but not on the `CreateOfferer`.

Alex the Entreprenerd (judge) commented:

> Have asked the Warden for a Coded POC.
> Have reached out to Seaport Devs to ask for advice as to whether the issue is valid.

Alex the Entreprenerd (judge) commented:

> Coded POC from the warden.

0xfoobar (Delegate) disagreed with severity and commented:

> Believe issue is valid, would love confirmation from Seaport devs if you have it.
> Believe this should be a quality Medium not High because CreateOfferer is tangential to the core protocol, would not cause any loss of user funds if abused,

> and could be easily replaced if DoS-ed. Appreciate the detailed work to dive into the workings of Seaport nonces.

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> Agree with impact from Sponsor. Temporarly marking as valid; however, we will not confirm until runnable POC is produced.

ladboy233 (warden) commented:

> Agree with sponsor's severity assessment!

> Fully runnable **POC** to show the nonce is out of sync.

> The Contract offerer internal nonce is tracked in this **line of code**.

> While the contract offerer nonce on Seaport side can be acquired by calling:

```
seaport.getContractOffererNonce(address(contractOfferer1));
```

> And indeed if we use `fulfillAvailableAdvancedOrder` to fulfill only one contract order and revert in other order, nonce is still incremented.

```
uint256 seaport_contrat_offer_1_nonce = seaport.getContract
   console2.log(seaport_contrat_offer_1_nonce); // the nonc
   uint256 oldNonce = contractOfferer1.nonce(); // the none
   console2.log(oldNonce);
```

> A more detailed explanation is **here**.

Alex the Entreprenerd (judge) commented:

> Logs

```
Logs:
  mockERC721.ownerOf(1) 0x7FA9385bE102ac3EAc297483Dd6233D62b3e14
  mockERC721.ownerOf(2) 0x7FA9385bE102ac3EAc297483Dd6233D62b3e14
  setRevert true
```

```
mockERC721.ownerOf(1) 0x7FA9385bE102ac3EAc297483Dd6233D62b3e14
mockERC721.ownerOf(2) 0x7FA9385bE102ac3EAc297483Dd6233D62b3e14
generateOrder
reverting
generateOrder
Sent ETH to SEAPORT true
fulfilled true
1
0
mockERC721.ownerOf(1) 0x7FA9385bE102ac3EAc297483Dd6233D62b3e14
mockERC721.ownerOf(2) 0xa0Cb889707d426A7A386870A03bc70d1b06975
```

> As shown by the logs, the Token 1 is not transfered, but the nonce is consumed.

POC

Low Risk and Non-Critical Issues

For this audit, 10 reports were submitted by wardens detailing low risk and non-critical issues. The **report highlighted below** by **DadeKuma** received the top score from the judge.

The following wardens also submitted reports: **p0wd3r**, **gkrastenov**, **Fulum**, **sces60107**, **Brenzee**, **kodyvim**, **lsaudit**, **ladboy233**, *and* **lodelux**.

[01] `previewOrder` should not revert

The official **documentation** says that `previewOrder` should not revert, as it may cause issues to the caller:

> An optimal Seaport app should return an order with penalties when its previewOrder function is called with unacceptable minimumReceived and maximumSpent arrays, so that the caller can learn what the Seaport app expects. But it should revert when its generateOrder is called with unacceptable minimumReceived and maximumSpent arrays, so the function fails fast, gets skipped, and avoids wasting gas by leaving the validation to Seaport.

```
function previewOrder(address caller, address, SpentItem[] callc
    external
```

```
        view
        onlySeaport(caller)
        returns (SpentItem[] memory offer, ReceivedItem[] memory cor
    {
        if (context.length != 160) revert Errors.InvalidContextLengt
        (offer, consideration) = Helpers.processSpentItems(minimumRe
    }
```

https://github.com/code-423n4/2023-09-delegate/blob/main/src/CreateOfferer.sol#L176-L184

[02] Withdraw should revert with non-supported `delegationType`

There seems to be no way to create a DelegateToken that is not ERC721/ERC20/ERC1155, but due to the fact that `DelegationType` supports also `ALL` and `CONTRACT`:

```
    enum DelegationType {
        NONE,
        ALL,
        CONTRACT,
        ERC721,
        ERC20,
        ERC1155
    }
```

And there are multiple ways to create DelegateToken (e.g. `DelegateToken.create` and `CreateOfferer.transferFrom`, but more contracts could be created in the future), it would be safer to revert the transaction on withdraw to avoid burning unsupported tokens:

```
    function withdraw(uint256 delegateTokenId) external nonReentrant
        ...
        } else if (delegationType == IDelegateRegistry.DelegationTyp
            uint256 erc1155UnderlyingAmount = StorageHelpers.readUnc
            StorageHelpers.writeUnderlyingAmount(delegateTokenInfo,
            uint256 erc11551UnderlyingTokenId = RegistryHelpers.loac
            RegistryHelpers.decrementERC1155(
```

```
                delegateRegistry, registryHash, delegateTokenHolder,
            );
            StorageHelpers.burnPrincipal(principalToken, principalBu
            IERC1155(underlyingContract).safeTransferFrom(address(th
        }
    /* @audit Should revert here to avoid burning a not supported de
        else {
            revert Errors.InvalidTokenType(delegationType);
        }
    */
    }
```

https://github.com/code-423n4/2023-09-delegate/blob/main/src/DelegateToken.sol#L353-L386

[03] Lack of `data` on flashloan could make some ERC1155 unusable

When doing an ERC1155 flashloan, the data parameter is always empty:

```
    IERC1155(info.tokenContract).safeTransferFrom(address(this), inf
```

https://github.com/code-423n4/2023-09-delegate/blob/main/src/DelegateToken.sol#L406

Some collections might need this parameter to be usable. This is an **example** of such case:

1. The sender transfers ERC1155 from the OpenSea contract to the ApeGang contract
2. In the same transaction the ApeGang's onERC1155BatchReceived hook is called
3. This allows the ApeGang to react to the token being received
4. The data includes merkle proofs that are needed to validate the transaction

Consider adding a `flashloanData` parameter to `Structs.FlashInfo` and pass it while transfering.

[04] Using `delegatecall` inside a loop may cause issues with `payable` functions

If one of the `delegatecall` consumes part of the `msg.value`, other calls might fail, if they expect the full `msg.value`. Consider using a different design, or fully document this decision to avoid potential issues.

```
function multicall(bytes[] calldata data) external payable overr
    results = new bytes[](data.length);
    bool success;
    unchecked {
        for (uint256 i = 0; i < data.length; ++i) {
            //slither-disable-next-line calls-loop,delegatecall-
            (success, results[i]) = address(this).delegatecall(d
            if (!success) revert MulticallFailed();
        }
    }
}
```

https://github.com/delegatexyz/delegate-registry/blob/6d1254de793ccc40134f9bec0b7cb3d9c3632bc1/src/DelegateRegistry.sol#L37

[05] `CreateOfferer` uses a custom context implementation instead of an existing SIP

It is suggested to use an existing SIP implementation instead of creating a new standard from scratch, which might be prone to errors:

```
Structs.Context memory decodedContext = abi.decode(context, (Str
```

https://github.com/code-423n4/2023-09-delegate/blob/main/src/CreateOfferer.sol#L59

Please note that the contract needs to be SIP compliant before it's possible to implement SIP-7, as it requires SIP-5 and SIP-6.

The issue describing non-compliance is described here: `#280` .

0xfoobar (Delegate) commented:

> Useful QA report.

Alex the Entreprenerd (judge) commented:

> **[01] previewOrder should not revert**
> Low
>
> **[02] Withdraw should revert with a not supported delegationType**
> Refactoring
>
> **[03] Lack of data on flashloan could make some ERC1155 unusable**
> Low
>
> **[04] Using delegatecall inside a loop may cause issues with payable functions**
> Refactoring
>
> **[05] CreateOfferer uses a custom context implementation instead of an existing SIP**
> Low
>
> The following downgraded submissions from the warden were also considered in scoring:
>
> - Seaport orders will not work with USDT
> - DelegateToken is not EIP-721 compliant
> - Rebasing tokens remain permanently locked inside DelegateToken
> - ETH can be permanently locked during a flashloan
> - Principal token can be permanently locked
> - CreateOfferer is not SIP-compliant, which can cause integration issues with third parties
>
> Total: 6+ Low and 2 Refactoring

> By far the best submission, great work!

0xfoobar (Delegate) acknowledged and commented:

> **[01] previewOrder should not revert**
> The quoted documentation actually says it can revert, we're not doing order penalties here just straightforward fulfillment.

> **[02] Withdraw should revert with non-supported delegationType**
> It does.

> **[03] Lack of data on flashloan could make some ERC1155 unusable**
> Acknowledged, we won't be transferring directly to staking contracts with merkle roots so this is fine.

> **[04] Using delegatecall inside a loop may cause issues with payable functions**
> But here it does not, we can see in the quoted code that it's not looping over `msg.value`.

> **[05] CreateOfferer uses a custom context implementation instead of an existing SIP**
> Acknowledged.

Gas Optimizations

For this audit, 2 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **Sathish9098** received the top score from the judge.

The following warden also submitted a report: Baki.

[G-01] Structs can be packed into fewer storage slots

Saves `6000 GAS , 3 SLOT`

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.

`expiry` can be uint96 instead of `uint256` : Saves `2000 GAS` , `1 SLOT`

https://github.com/code-423n4/2023-09-delegate/blob/a6dbac8068760ee4fc5bababb57e3fe79e5eeb2e/src/libraries/DelegateTokenLib.sol#L20-L29

A `uint96` can store values from `0 to 2^96 - 1` , which is a very large range. However, it's important to note that Ethereum's `block.timestamp` is a `Unix timestamp` , which represents time in seconds.

A `uint96` would overflow after approximately `2,508,149,904,626,209 years` when storing time in seconds. This is an extremely long time frame, and it's highly unlikely that Ethereum or any blockchain system will remain unchanged for such an extended period.

```
FILE: 2023-09-delegate/src/libraries/DelegateTokenLib.sol

20: struct DelegateInfo {
21:         address principalHolder;
22:         IDelegateRegistry.DelegationType tokenType;
23:         address delegateHolder;
24:         uint256 amount;
25:         address tokenContract;
+ 28:           uint96 expiry;
26:         uint256 tokenId;
27:         bytes32 rights;
- 28:           uint256 expiry;
29:     }
```

`signerSalt` , `expiryLength` can be `uint128` instead of `uint256` : Saves `4000 GAS` , `2 SLOT`

https://github.com/code-423n4/2023-09-delegate/blob/a6dbac8068760ee4fc5bababb57e3fe79e5eeb2e/src/libraries/CreateOffererLib.sol#L89-L105

In many blockchain protocols, the usage of `salt` often involves values within the range of `uint32` , as they provide a sufficient numeric space for generating unique

salts. Therefore, adopting a uint32 data type for the signerSalt field is a reasonable choice, as it aligns with common practices and conserves storage resources.

For the `expiryLength` field, which is intended to store the `duration` or `length` of an `expiration period`, using `uint128` is more than adequate. This choice allows for a vast range of possible expiration lengths, accommodating a wide spectrum of use cases without incurring unnecessary storage overhead

```
FILE: 2023-09-delegate/src/libraries/CreateOffererLib.sol

89: struct Context {
90:         bytes32 rights;
+ 91:         uint128 signerSalt;
+ 92:         uint128 expiryLength;
- 91:         uint256 signerSalt;
- 92:         uint256 expiryLength;
93:         CreateOffererEnums.ExpiryType expiryType;
94:         CreateOffererEnums.TargetToken targetToken;
95:     }


98: struct Order {
99:         bytes32 rights;
- 100:         uint256 expiryLength;
- 101:         uint256 signerSalt;
+ 100:         uint128 expiryLength;
+ 101:         uint128 signerSalt;
102:         address tokenContract;
103:         CreateOffererEnums.ExpiryType expiryType;
104:         CreateOffererEnums.TargetToken targetToken;
105:     }
```

[G-02] Remove or replace unused state variables

Saves `20000 GAS`

Saves a storage slot. If the variable is assigned a non-zero value, saves `Gsset (20000 gas)`. If it's assigned a zero value, saves `Gsreset (2900 gas)`. If the variable remains unassigned, there is no gas savings unless the variable is public, in which case the compiler-generated non-payable getter deployment cost is saved. If

the state variable is overriding an interface's public function, mark the variable as constant or immutable so that it does not use a storage slot

```
FILE: delegate-registry/src/DelegateRegistry.sol

18: mapping(bytes32 delegationHash => bytes32[5] delegationStora
```

https://github.com/delegatexyz/delegate-registry/blob/6d1254de793ccc40134f9bec0b7cb3d9c3632bc1/src/DelegateRegistry.sol#L18

[G-03] State variables should be cached

Saves `300 GAS , 3 SLOD`

Caching of a state variable replaces each Gwarmaccess (100 gas) with a 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.

`nonce.value , receivers.targetTokenReceiver , receivers.fulfiller` should be cached : Saves `300 GAS , 3 SLOD`

https://github.com/code-423n4/2023-09-delegate/blob/a6dbac8068760ee4fc5bababb57e3fe79e5eeb2e/src/libraries/CreateOffererLib.sol#L185

```
FILE: 2023-09-delegate/src/libraries/CreateOffererLib.sol

+ uint256 value_ = nonce.value ;
- 185: if (nonce.value != contractNonce) revert CreateOffererErr
+ 185: if (value_ != contractNonce) revert CreateOffererErrors.I


+ address targetTokenReceiver_ = receivers.targetTokenReceiver ;
+ address fulfiller_ = receivers.fulfiller;
289: //slither-disable-start timestamp
300:         if (
             keccak256(
```

```
                abi.encode(
                    IDelegateTokenStructs.DelegateInfo({
                        tokenType: tokenType,
-                        principalHolder: decodedContext.target
  ? receivers.targetTokenReceiver : receivers.fulfiller,
+                        principalHolder: decodedContext.target
  ? targetTokenReceiver_  : fulfiller_ ,
-                       delegateHolder: decodedContext.targetTo
+                       delegateHolder: decodedContext.targetTo
                        expiry: CreateOffererHelpers.calculateEx
                        rights: decodedContext.rights,
                        tokenContract: consideration.token,
                        tokenId: (tokenType != IDelegateRegistry
                        amount: (tokenType != IDelegateRegistry.
                    })
                )
            ) != keccak256(abi.encode(IDelegateToken(delegateTok
        ) revert CreateOffererErrors.DelegateInfoInvariant();
```

[G-04] Use assembly for loops to save gas

Saves `2450 GAS` for every iteration from `7 instances`

Assembly is more gas efficient for loops. Saves minimum `350 GAS` per iteration as per remix gas checks.

```
FILE: Breadcrumbsdelegate-registry/src/DelegateRegistry.sol

- 35: for (uint256 i = 0; i < data.length; ++i) {
-                //slither-disable-next-line calls-loop,delegate
-                (success, results[i]) = address(this).delegatec
-                if (!success) revert MulticallFailed();
-            }

+ assembly {
+    // Load the length of the data array
+    let dataSize := mload(data)
+
+    // Initialize the results array
+    let results := mload(0x40)
+    mstore(results, dataSize)
+
+    // Initialize a counter (i) to zero
```

```
+    let i := 0
+
+    for { } lt(i, dataSize) { } {
+        // Start loop
+
+        // Load the next calldata from the data array
+        let calldataPtr := add(add(data, 0x20), mul(i, 0x20))
+        let calldataSize := mload(calldataPtr)
+
+        // Perform delegatecall
+        let success := delegatecall(gas(), address(), add(callc
+
+        // Store the result and check for success
+        if iszero(success) {
+            // Revert if delegatecall fails
+            revert(0, 0)
+        }
+        mstore(add(results, mul(i, 0x20)), success)
+
+        // Increment the counter
+        i := add(i, 1)
+
+        // End loop
+    }
+
+    // results contains the success status of each delegatecall
+
+ }
```

```
275:  for (uint256 i = 0; i < hashes.length; ++i) {
```

```
312: for (uint256 i = 0; i < length; ++i) {
                tempLocation = locations[i];
                assembly {
                    tempValue := sload(tempLocation)
                }
                contents[i] = tempValue;
            }
```

```
386: for (uint256 i = 0; i < hashesLength; ++i) {
                hash = hashes[i];
                if (_invalidFrom(_loadFrom(Hashes.location(hash)
                filteredHashes[count++] = hash;
            }
```

```
393:  for (uint256 i = 0; i < count; ++i) {
```

```
                hash = filteredHashes[i];
                location = Hashes.location(hash);
                (address from, address to, address contract_) =
                delegations_[i] = Delegation({
                    type_: Hashes.decodeType(hash),
                    to: to,
                    from: from,
                    rights: _loadDelegationBytes32(location, Sto
                    amount: _loadDelegationUint(location, Storag
                    contract_: contract_,
                    tokenId: _loadDelegationUint(location, Stora
                });
            }

417:        for (uint256 i = 0; i < hashesLength; ++i) {
                hash = hashes[i];
                if (_invalidFrom(_loadFrom(Hashes.location(hash)
                filteredHashes[count++] = hash;
            }

423:        for (uint256 i = 0; i < count; ++i) {
                validHashes[i] = filteredHashes[i];
            }
```

https://github.com/delegatexyz/delegate-registry/blob/6d1254de793ccc40134f9bec0b7cb3d9c3632bc1/src/DelegateRegistry.sol#L423-L425

[G-05] Don’t initialize default values to variables to reduce gas

Saves `13 GAS` for local variable and `2000 GAS` for state variable

```
FILE: Breadcrumbsdelegate-registry/src/DelegateRegistry.sol

35: for (uint256 i = 0; i < data.length; ++i) {
275: for (uint256 i = 0; i < hashes.length; ++i) {
312: for (uint256 i = 0; i < length; ++i) {
386: for (uint256 i = 0; i < hashesLength; ++i) {
393: for (uint256 i = 0; i < count; ++i) {
417: for (uint256 i = 0; i < hashesLength; ++i) {
423: for (uint256 i = 0; i < count; ++i) {
```

https://github.com/delegatexyz/delegate-registry/blob/6d1254de793ccc40134f9bec0b7cb3d9c3632bc1/src/DelegateRegistry.sol#L35

[G-06] Use constants instead of type(uintX).max to avoid calculating every time

```
FILE: delegate-registry/src/DelegateRegistry.sol

213:     ? type(uint256).max
215: if (!Ops.or(rights == "", amount == type(uint256).max)) {
217: ? type(uint256).max
232: ? type(uint256).max
234: if (!Ops.or(rights == "", amount == type(uint256).max)) {
236: ? type(uint256).max
372: uint256 cleanUpper12Bytes = type(uint256).max << 160;
```

https://github.com/delegatexyz/delegate-registry/blob/6d1254de793ccc40134f9bec0b7cb3d9c3632bc1/src/DelegateRegistry.sol#L213

[G-07] For same condition checks use modifiers

```
File: delegate-registry/src/DelegateRegistry.sol

56: } else if (loadedFrom == msg.sender) {
75: } else if (loadedFrom == msg.sender) {
85: } else if (loadedFrom == msg.sender) {
115:  } else if (loadedFrom == msg.sender) {
118: } else if (loadedFrom == msg.sender) {
140: } else if (loadedFrom == msg.sender) {
143: } else if (loadedFrom == msg.sender) {
```

https://github.com/delegatexyz/delegate-registry/blob/6d1254de793ccc40134f9bec0b7cb3d9c3632bc1/src/DelegateRegistry.sol#L75C9-L75C47

[G-08] Declare the variables outside the loop

Per iterations saves `26 GAS`

```
FILE: delegate-registry/src/DelegateRegistry.sol

+                bytes32 location ;
+                address from ;
 for (uint256 i = 0; i < hashes.length; ++i) {
-                bytes32 location = Hashes.location(hashes[i]);
-                address from = _loadFrom(location);
+                location = Hashes.location(hashes[i]);
+                from = _loadFrom(location);
                 if (_invalidFrom(from)) {
                     delegations_[i] = Delegation({type_: Delegat
                 } else {
                     (, address to, address contract_) = _loadDel
                     delegations_[i] = Delegation({
                         type_: Hashes.decodeType(hashes[i]),
                         to: to,
                         from: from,
                         rights: _loadDelegationBytes32(location,
                         amount: _loadDelegationUint(location, St
                         contract_: contract_,
                         tokenId: _loadDelegationUint(location, 
                     });
```

Alex the Entreprenerd (judge) commented:

> Very good work with the first 2 findings!

0xfoobar (Delegate) confirmed and commented:

> Useful findings on the gas stuff. Not sure the refactoring is worth the loss of struct ordering here but will think it over.

Audit Analysis

For this audit, 4 analysis reports were submitted by wardens. An analysis report examines the codebase as a whole, providing observations and advice on such

topics as architecture, mechanism, or approach. The report highlighted below by **pfapostol** received the top score from the judge.

The following wardens also submitted reports: DadeKuma, m4ttm, *and* BanditxOx.

Approach taken in evaluating the codebase

I first explored the scope of audit. I discovered that the project can be divided into 2 independent parts: `Delegate Registry` and `Delegate Marketplace`. I carried out all subsequent stages separately for each of this parts, and then analyzed the correctness of their interaction.

Test coverage

`Delegate Registry`:

Test coverage is 100% for most audit files. In this regard, I decided to concentrate on finding logical errors, since simple errors (errors due to typos, incorrect statements) should be excluded by tests.

File	% Lines	% Statements	% Branches	% Funcs
src/DelegateRegistry.sol	100.00% (175/175)	100.00% (219/219)	98.78% (81/82)	100.00% (33/33)
src/libraries/RegistryHashes.sol	100.00% (12/12)	100.00% (12/12)	100.00% (0/0)	100.00% (12/12)
src/libraries/RegistryOps.sol	66.67% (2/3)	66.67% (2/3)	100.00% (0/0)	66.67% (2/3)
src/libraries/RegistryStorage.sol	100.00% (6/6)	100.00% (6/6)	100.00% (0/0)	100.00% (3/3)

`Delegate Marketplace`:

File	% Lines	% Statements	% Branches	% Funcs
src/CreateOfferer.sol	90.70% (39/43)	92.00% (46/50)	77.27% (17/22)	100.00% (8/8)
src/DelegateToken.sol	88.55% (147/166)	90.28% (195/216)	80.43% (37/46)	89.66% (26/29)

File	% Lines	% Statements	% Branches	% Funcs
src/PrincipalToken.sol	100.00% (14/14)	100.00% (17/17)	100.00% (4/4)	100.00% (5/5)
src/libraries/CreateOffererLib.sol	95.24% (40/42)	95.38% (62/65)	69.23% (18/26)	100.00% (9/9)
src/libraries/DelegateTokenLib.sol	88.89% (8/9)	90.48% (19/21)	75.00% (6/8)	100.00% (5/5)
src/libraries/DelegateTokenRegistryHelpers.sol	100.00% (57/57)	100.00% (87/87)	100.00% (26/26)	100.00% (21/21)
src/libraries/DelegateTokenStorageHelpers.sol	91.67% (44/48)	92.11% (70/76)	80.77% (21/26)	100.00% (21/21)
src/libraries/DelegateTokenTransferHelpers.sol	88.24% (30/34)	87.80% (36/41)	80.77% (21/26)	100.00% (9/9)

Code review

I studied the `Delegate Registry` code starting with the libraries, and also starting from the lowest level functions, moving to the top level functions. Having built a general understanding of what each of the functions does, I formed an idea of how the Registry works and built general diagrams.

Packed delegation data:

Structure of packed delegation			
slot	32 bytes		
0	empty 4 bytes	first 8 bytes of contract	cleaned `from` address
1	last 12 bytes of contract		cleaned `to` address
2	Rights		
3	TokenId (for ERC721 (ERC1155) delegation)		
4	Amount (for ERC20 (ERC1155) delegation)		

Important external interfaces:

1. delegateAll - Delegates the entire wallet

2. delegateContract - Delegates the right to use the contract
3. delegateERC721 - Delegates the right to use a specific contract token
4. delegateERC20 - Delegates the right to use a certain amount of a token of a certain contract
5. delegateERC1155 - Delegates the right to use a certain amount of a certain token of a certain contract

There are also 5 functions to check the correctness of the delegation

Details on each function and hashing schemes
`Delegate Marketplace`:

`CreateOfferer` is a separate part of the marketplace that guarantees interaction with the seaport.

Important external interfaces:

1. create - Create `DelegateToken` and `PrincipalToken` tokens. Transfer one of token types to contract. Delegate to `delegateHolder` . mint principal token.
2. extend - Extend the expiration time for an existing `DelegateToken` . Called by `PrincipalToken` owner.
3. rescind - Return the `DelegateToken` to the `PrincipalToken` holder early. Called by `DelegateToken` holder or after the `DelegateToken` has expired, anyone can call this method. this does not release the spot asset from escrow, it merely cancels out the `DelegateToken` .
4. withdraw - burn the `PrincipalToken` and claim the spot asset from escrow. Called by the `PrincipalToken` owner. `PrincipalToken` owner can authorize others to call this on their behalf, and if `PrincipalToken` owner also owns the `DelegateToken` then they can skip calling rescind and go straight to withdraw

Details for each function

Mechanism review

The contract consists of 2 parts, one part is a storage of delegation hashes, and the other part is ERC721 compatible tokens that reflect the ownership of the delegation.

`Delegate Registry` : uses hashing to compactly store the delegation. And also hash functions for calculating a unique location in storage. It also contains functions that check the hash based on the location and `from` address.

`Delegate Token` : Deposits all assets, in return issues an ERC721 token, which confirms the ownership of the delegation, for a certain period of time.

`PrincipalToken` : Depends on `Delegate Token` , cannot be called on its own. It is an ERC721 token that confirms the right to claim deposited assets after expiration.

`CreateOfferer` : Integration with seaport as specified in documentation. When selling, the asset turns into a `Delegate Token` and is assigned to the buyer, the seller receives a `PrincipalToken` .

`Delegate Token` in its work relies entirely on `Delegate Registry` , which must reliably guarantee the authenticity and confirmation of the delegation.

Codebase quality analysis

In general, the quality of the code base is quite high. The huge number of comments in NatSpec makes it very easy to determine what a particular function is intended for.

The downside is the use of assembler for gas optimization, which is not comparable to the damage it causes to code readability.

Centralization risks

There is no risk of centralization since all rights are divided between the `Delegate Token` and the `PrincipalToken` . The only exception is `CreateOfferer` , which relies on the `seaport` address, which is immutable, but it is possible that the contract address will change in the future, it would be useful to add a function that allows you to change the address if necessary

Systemic risks

The contract is used to delegate all types of tokens (`ERC20` , `ERC721` , `ERC1155`), but does not take into account that some tokens do not follow the standards.

Contracts are programmed for version ^0.8.21, by default the compiler will use version 0.8.21, which is very recent and may contain undetected vulnerabilities, as well as compatibility problems with different L2 chains.

New insights and learning from this audit

I learned about `CreateOfferer` seaport integration, all other concepts were well known to me.

Time spent
33 hours

Alex the Entreprenerd (judge) commented:

> Imo proper way to discuss coverage
> +
> Interesting charts for packing and logic on delegation

Disclosures

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

C4 Audits incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Audit 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