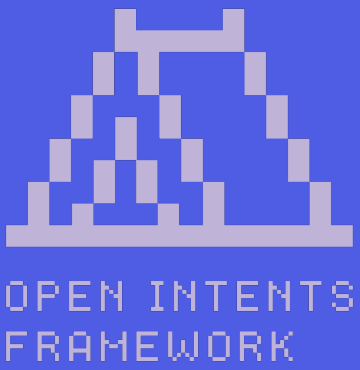
OpenZeppelin | security

OIF Contracts Diff Audit



October 30, 2025

Table of Contents

Summary

Type	Libraries
Timeline	From 2025-09-22 To 2025-10-03
Languages	Solidity

Total Issues	8 (7 resolved, 1 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	1 (0 resolved, 1 partially resolved)
Low Severity Issues	3 (3 resolved)
Notes & Additional Information	4 (4 resolved)

Scope

OpenZeppelin performed a diff-audit of the openintentsframework/oif-contracts repository, between base commit 7153291 and target commit eafdaa8. This diff highlights all the changes made between the two commits.

In scope were the following files:

```
src
├── input
│   ├── InputSettlerBase.sol
│   ├── InputSettlerPurchase.sol
│   ├── compact
│   │   └── InputSettlerCompact.sol
│   ├── escrow
│   │   ├── InputSettlerEscrow.sol
│   │   └── Permit2WitnessType.sol
│   └── types
│       ├── MandateOutputType.sol
│       └── StandardOrderType.sol
├── integrations
│   ├── CatsMulticallHandler.sol
│   └── oracles
│       └── hyperlane
│           └── HyperlaneOracle.sol
├── interfaces
│   ├── IAttester.sol
│   ├── IInputCallback.sol
│   ├── IInputOracle.sol
│   ├── IInputSettlerCompact.sol
│   ├── IInputSettlerEscrow.sol
│   └── IOutputCallback.sol
├── libs
│   └── AssemblyLib.sol
├── oracles
│   └── BaseInputOracle.sol
└── output
    ├── OutputSettlerBase.sol
    └── simple
        ├── FillerDataLib.sol
        ├── FulfilmentLib.sol
        └── OutputSettlerSimple.sol
```

> ***Update:*** *The final state of the audited codebase, including all implemented resolutions, is reflected in commit 222989b.*

System Overview

The Open Intents Framework (OIF) is a decentralized protocol designed for expressing and settling user intents across different blockchain networks. With intents, instead of performing a transaction themselves, users specify what they want (the intent), and let specialized actors handle the rest. The OIF architecture separates the expression of an intent on a source chain from its fulfillment on a destination chain. This is achieved through a system of Input Settlers, which lock user assets, and Output Settlers, which are triggered by solvers who fulfill the intent's conditions. A cross-chain oracle layer is responsible for verifying and attesting to the fulfillment of these conditions.

The changes under review consist primarily of refactoring and the migration from Solady to OpenZeppelin libraries for signature-checking, ERC-20, and EIP-712 functionality. The diffs were reviewed against the new `v5.5.0-rc.0` release of OpenZeppelin Contracts. The most notable changes are summarized below:

ERC-20 Functionality

In Solady, safe ERC-20 transfers and approvals revert if the call succeeds, but the token contract does not return a value. The OpenZeppelin Contracts library instead treats such calls as successful, accommodating non-compliant tokens like USDT that omit return values in `transfer`, `transferFrom` or `approve`.

When querying token balances, the OpenZeppelin Contracts library enforces stricter behavior: calling `balanceOf` on a non-existent token will revert. In contrast, Solady's implementation is more permissive and returns 0 in the same situation.

Support for Native Token Outputs

The refactor introduces support for orders with outputs denominated in the native token. The order filler must attach sufficient native tokens to fulfill the order. Any excess amount is refunded at the end of execution.

`Ownable` Changes

In Solady, the `Ownable` mixin provides both single-step and 2-step ownership transfers. In contrast, the OpenZeppelin `Ownable` contract only supports single-step ownership transfers. In the diff under review, the `ChainMap` contract updates its dependency from the Solady `Ownable` mixin to the OpenZeppelin `Ownable` contract.

Medium Severity

M-01 Multi-Output Orders Can Be Manipulated

Orders can include multiple outputs, all of which must be filled for input escrow funds to be released.

To avoid conflicts, the filler of the first output is treated as the order owner, gaining substantial control over order flow. This introduces two risks:

- **Denial of Service:** The first filler may stall the order by not completing other outputs, leaving user funds locked until cancellation or delaying fills to the last moment. Note that this attack is mostly impractical for the attacker, as they would either lose funds in the process or pay the "opportunity cost" for the locked funds.
- **Manipulation of Dutch Auction Orders:** If later outputs are priced via a Dutch auction, an attacker can claim the first output, wait for the auction price to fall, and secure a more favorable rate.

While defining ownership via the first output is a pragmatic choice, users should be cautioned about the risks of complex, multi-output orders. Consider encouraging users to use "exclusive" orders with trusted fillers. For multi-output Dutch auctions, consider recording the timestamp of the first fill and using it to calculate subsequent auction prices, reducing the artificial advantage of early claimers.

Update: *Partially Resolved in pull request #145. The OpenZeppelin Contracts team stated:*

> *The team considers that the protocol should remain open and users and solvers should be free to agree between themselves on the terms of orders to be opened/filled. That said, they should be aware of risks and security assumptions taken when using the contracts, so we updated the natspec of the contracts, explaining the risks associated and recommendations for users, solvers or anyone else integrating with the contracts (see here). In general, the recommendation is that users should not open orders where any output (other than the first) has a variable output amount (which is the case of the dutch auction) and should have the first output as the most valuable to increase the cost of a Denial of Service attack. From the solver perspective, they should be certain that they will be able to fill all of the outputs in time in order to fill the first output.*

> *Besides that, upon release of the contracts, we will release a robust documentation explaining all of the risks and checklist for solvers/integrators.*

Low Severity

L-01 Compact Format Signatures (64 bytes) Are No Longer Supported

The system relies on EIP-712 signatures to authorize external claimants or order purchases. The signature-verification logic was migrated from Solady's `SignatureCheckerLib` to OpenZeppelin's `SignatureChecker`, which introduces an important compatibility change:

- Solady supports both 64- and 65-byte signature formats.
- OpenZeppelin only supports the 65-byte format.

As a result, any compact (64-byte) signatures are now invalid. If backward compatibility with compact signatures is required, explicit support for the 64-byte format should be reintroduced.

Other Observed Differences (no security risk identified):

- OpenZeppelin leverages the `ECDSA` library, which provides a safer wrapper around the `ecrecover` precompile. It ensures both that signature recovery succeeds (non-zero recovered signer) and that the signature is not malleable. This check was missing in Solady, but malleability was not a risk in practice since an order can only be purchased or finalized once.

Update: *Resolved. Not an issue, the change was intentional. The OpenZeppelin Contracts team stated:*

> *This is a known impact and the team along with other partners has agreed that supporting only 65-byte format is fine and compatible with existing integrators.*

L-02 2-Step Ownership Transfer Is No Longer Supported

The `ChainMap` contract is an abstract contract designed to add chain-mapping functionality to oracle implementations. It restricts the `setChainMap` function to a privileged owner, who is

responsible for configuring the mapping between a protocol's chain identifier and a canonical chain ID.

The contract currently inherits from OpenZeppelin's `Ownable` contract, which facilitates a single-step ownership transfer. A call to the `transferOwnership` function immediately transfers the owner role to the provided address. If this address is incorrect, non-existent, or otherwise inaccessible, the ownership of the contract is irrevocably lost, preventing any future administrative actions. It is important to note that the `ChainMap` contract was previously using the `Ownable` implementation from the Solady library, which contained 2-step ownership transfer functionality, reducing the above-mentioned risks.

If such functionality is still of interest, consider moving from OpenZeppelin's `Ownable` contract to `Ownable2Step`.

Update: *Resolved. Not an issue, the change was intentional. The OpenZeppelin Contracts team stated:*

> *This is an intended change and the team does not want to support 2 step ownership change. Should not be considered an issue.*

L-03 Relationship Between `fillDeadline` and `expiryDeadline` Is Not Enforced

Within the `open` function of the `InputSettlerEscrow` contract, there are checks ensuring that the current timestamp is less than both `order.fillDeadline` and `order.expires` values. However, there are no checks enforcing the `fillDeadline` to be less than the `expiryDeadline`.

Since these deadlines are intended to follow a natural sequence, consider adding a check that `fillDeadline` < `expires`. This would serve as a useful sanity check, helping prevent user errors and ensuring the system behaves in a predictable manner.

Update: *Resolved in pull request #142.*

Notes & Additional Information

N-01 Unused or Redundant Code

Throughout the codebase, multiple instances of unused or redundant code were identified:

- The `EfficiencyLib` library is no longer used in the `InputSettlerBase`, `InputSettlerPurchase`, `InputSettlerCompact`, `InputSettlerEscrow`, and `CatsMulticallHandler` contracts. Consider removing it.
- The `InputSettlerCompact` and `InputSettlerEscrow` contracts use the return values of the `_domainName` and `_domainVersion` functions to instantiate the `EIP712` contract. Since these functions are not used elsewhere, their logic could be replaced with hardcoded string literals or constants, reducing contract size and simplifying the code.

Consider removing or refactoring any redundancies to improve the clarity and maintainability of the codebase.

Update: *Resolved in pull request 141. The OpenZeppelin Contracts team stated:*

> *This issue can be split in two. For the unused EfficiencyLib, this PR has been created: https://github.com/openintentsframework/oif-contracts/pull/141. Regarding the use of domain name and version functions, this is a required functionality to allow for other protocols to inherit from this contract and be able to override those values, as identified by some of our partners during the development pipeline.*

N-02 Inconsistent Handling of Dirty Bits in Token Address

The `InputSettlerEscrow` contract provides functionality for users to open orders by escrowing assets. The contract offers the `open` function for users to deposit assets directly, and the `openFor` function to allow a third party to open an order on a user's behalf using a signature. The `openFor` function supports various signature schemes such as Permit2 and ERC-3009, which results in different internal code paths to handle the token transfers.

An inconsistency exists in how the token address is handled across these different functions. The `open` function and the Permit2 flow within `openFor` both use the `validatedCleanAddress()` library function to revert if the upper 12 bits of the 256-bit input value are dirty. However, the code path for handling ERC-3009 signatures uses `fromIdentifier()`, which does not revert but only removes the dirty bytes.

While this does not currently pose a security risk, consider enforcing consistency and updating the ERC-3009 flow to also revert if the first 12 bytes within the 32-byte `token` value are dirty.

Update: *Resolved in pull request #140. The OpenZeppelin Contracts team stated:*

> *We've identified a couple of extra places where we used the `fromIdentifier()` function rather than `validatedCleanAddress()` on top of the one identified in the audit.*

N-03 Typographical Errors

Throughout the codebase, multiple instances of typographical errors were identified:

- In line 23 of `IInputOracle.sol`, the word "chucks" should say "chunks".
- In line 137 of `CatsMulticallHandler.sol`, the word "dedublication" should say "de-duplication".

Consider scanning the codebase using automated tooling and correcting all typographical errors.

Update: *Resolved in pull request #139. The OpenZeppelin Contracts team stated:*

> *We also modified other errors.*

N-04 Naming Issues

Throughout the codebase, multiple instances of naming issues were identified:

- The `_dutchAuctionSlope` function does not return the slope, but rather the current price of a dutch auction. Consider renaming the function to `_dutchAuctionPrice`.
- The `call` member of the `MandateOutput` struct overshadows the Solidity reserved keyword `call`, and is often used as `output.call`. Consider renaming this member to differentiate it from the reserved keyword `call`.

Consider addressing the aforementioned naming issues to improve the clarity and maintainability of the codebase.

Update: *Resolved in pull request #138.*

Conclusion

The audited diff introduces changes in an incremental and methodical manner. Most updates focus on code quality, such as replacing untyped data with structs and migrating from Solady dependencies to OpenZeppelin contracts. Several improvements to documentation have also been made.

Overall, the codebase was found to be elegant and well-structured, with only a few issues identified around subtle behavioral differences and best practices. The codebase appears to be in good condition, though regular audits are recommended as more substantial functionality is introduced.