Eco Routes
Security Review

Cantina Managed review by:

0xRajeev, Lead Security Researcher
Phaze, Lead Security Researcher

March 12, 2025

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Eco enables apps to unlock stablecoin liquidity from any connected chain and give users the simplest onchain experience.

From Feb 5th to Feb 15th the Cantina team conducted a review of eco-routes on commit hash 69809122. The team identified a total of **25** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	1	1	0
High Risk	3	3	0
Medium Risk	6	4	2
Low Risk	10	6	4
Gas Optimizations	0	0	0
Informational	5	3	2
Total	**25**	**17**	**8**

Note: *After the fix review at commit 2e7bc2f as referenced in all the report findings, there was a bug fix which required further commits. The Cantina Managed team worked with the Eco team to briefly review the bug fix. The final commit which was reviewed is d7978544.*

3 Findings

3.1 Critical Risk

3.1.1 An incorrect predicate operator allows anyone to prevent fillers from withdrawing rewards by overfunding intent vaults

Severity: Critical Risk

Context: IntentVault.sol#L69-L72

Summary: Anyone can prevent fillers from withdrawing rewards by overfunding intent vaults because an incorrect predicate `amount < balance` is used in the vault logic.

Finding Description: Intent vaults are created using deterministic addresses by `IntentSource` for each intent. The vault handles token and native currency transfers and self-destructs after distributing rewards. Intent vaults are expected to be funded by intent creators. The current logic allows intent vaults to be overfunded with either the source chain's native token or any other ERC20 reward token (i.e. more than what's specified in an intent's `reward.nativeValue` or `reward.token[i].amount`).

However, the intent vault logic for withdrawals uses an incorrect predicate `amount < balance` in a reverting conditional check for `InsufficientTokenBalance` while verifying sufficient balance of reward tokens.

Illustrative Scenario:

1. Intent creator Carol publishes an intent on Arbitrum to be filled on Base. The intent involves sending 100 USDC to Carol's address C on Base. Carol's intent vault on Arbitrum is funded with 101 USDC to reward the potential filler with an additional 1 USDC for the fulfillment.
2. Filler Frank fulfills this intent on Base by sending 100 USDC to Carol's address C on Base. Frank choses to prove via a storage proof on Arbitrum.
3. Carol observes the intent being filled on Base in step (2) and immediately deposits 1 USDC to her intent vault, which will now have 102 USDC.
4. When Frank attempts to withdraw his rewards on Arbitrum, the balance on the intent vault is 102 USDC but the reward amount is 101 USDC. The withdrawal logic reverts as described above.
5. Frank cannot withdraw his deserved rewards on Arbitrum and loses his 101 USDC used for intent fulfillment on Base.
6. After the intent deadline, Carol withdraws her entire 102 USDC balance from the intent vault.
7. Carol got Frank to fulfill her intent using his own funds which are not refunded/rewarded.

Impact Explanation: High, because this prevents fillers from withdrawing their rewards on the source/origin chain even after filling intents on the destination chain, leading to complete loss of their funds used for filling intents. Vault reward funds go back to the intent creator after the deadline.

Likelihood Explanation: High, because this allows anyone, including a malicious intent creator who is especially motivated, to deposit any amount of any reward token in excess of `reward.tokens.amount` to force this check to revert during reward withdrawals by fillers. This applies to all intents that have any non-native ERC20 reward tokens, whose likelihood is reasonably high for intent usecases. Fillers can check that intents are funded via `isIntentFunded()` but `withdrawRewards()` will revert during withdrawal.

Recommendation: Consider:

1. Flipping the predicate to `amount > balance` in `IntentVault.constructor()`.
2. An alternative design where this check can potentially be removed to allow solvers to fill partially funded intents and be rewarded for that. For e.g.: some solvers may be fine with filling an intent for USDC 1.000000 (partially funded) instead of the stated reward of USDC 1.000001, and if they fail to check the fully funded status (USDC 1.000000 vs USDC 1.000001) before filling that intent then the current design will penalize them with 0 rewards because of this check.

Eco: Fixed in commit 2e7bc2f.

Cantina Managed: Reviewed that commit 2e7bc2f fixes this issue with the below logic:

```
function _processRewardTokens(
    Reward memory reward,
    address claimant
) internal {
    uint256 rewardsLength = reward.tokens.length;

    for (uint256 i; i < rewardsLength; ++i) {
        address token = reward.tokens[i].token;
        uint256 amount = reward.tokens[i].amount;
        uint256 balance = IERC20(token).balanceOf(address(this));

        if (claimant == reward.creator || balance < amount) {
            if (claimant != reward.creator) {
                emit RewardTransferFailed(token, claimant, amount);
            }
            if (balance > 0) {
                _tryTransfer(token, claimant, balance);
            }
        } else {
            _tryTransfer(token, claimant, amount);

            // Return excess balance to creator
            if (balance > amount) {
                _tryTransfer(token, reward.creator, balance - amount);
            }
        }
    }
}
```

where if `balance > amount` (else branch) then `amount` is transferred to `claimant` and excess balance is refunded to `creator`.

3.2 High Risk

3.2.1 ERC-7683 intents can never be filled because of an incorrectly encoded contract address

Severity: High Risk

Context: Eco7683OriginSettler.sol#L186-L190, Eco7683OriginSettler.sol#L280-L282, ERC7683.sol#L96-L101

Summary: ERC-7683 intents can never be filled in the protocol because an incorrect contract address is used in `FillInstruction.destinationSettler`.

Finding Description: Eco adds ERC-7683 compliant wrappers to allow ERC-7683 aware fillers to interoperate with the protocol. These wrappers are implemented in `Eco7683OriginSettler.sol` on the origin/source chains and `Eco7683DestinationSettler.sol` on the destination chains. `Eco7683OriginSettler` implements `open()`, `openFor()`, `resolve()` and `resolveFor()` functions and `Eco7683DestinationSettler` implements `fill()` function per ERC-7683.

However, while resolving an `OnchainCrossChainOrder` to a `ResolvedCrossChainOrder` in `resolve()` and `resolveFor()`, Eco's `inbox` contract address is incorrectly used instead of `bytes20(uint160(Eco7683DestinationSettler))` for `struct FillInstruction.destinationSettler` that is expected to be "The contract address that the order is meant to be filled on" for ERC-7683 intents.

Impact Explanation: Medium, because fillers cannot use the ERC-7683 wrappers and instead have to rely on Eco's native intent logic.

Likelihood Explanation: High, because an ERC-7683 aware filler picking up such an intent will attempt to call the corresponding ERC-7683 `fill()` on Eco's `inbox` contract which will always revert because it does not implement the ERC-7683 `fill()` function.

Recommendation: Consider using `bytes20(uint160(Eco7683DestinationSettler))` for `struct FillInstruction.destinationSettler` in `resolve()` and `resolveFor()` by appropriately making that address accessible in `Eco7683OriginSettler.sol`.

Eco: To solve this we did a bit of restructuring - the inbox is now the Eco7683DestinationSettler. the latter is now an abstract contract whose functionality is inherited by the inbox. See commit 2e7bc2f.

Cantina Managed: Reviewed that commit 2e7bc2f fixes this issue with their above mentioned restructuring.

3.2.2 Malicious reward token can block legitimate reward claims

Severity: High Risk

Context: IntentVault.sol#L52-L83

Summary: The IntentVault contract attempts to transfer all reward tokens atomically when an intent is fulfilled. A malicious token in the reward list can prevent the transfer of legitimate reward tokens, allowing intent creators to trick solvers with honeypot intents containing both legitimate and malicious reward tokens.

Description: When an intent is fulfilled, the vault tries to transfer all reward tokens to the claimant in a single transaction. If any token transfer fails (e.g., due to a malicious token that reverts on transfer), the entire claim transaction will revert. An intent creator can exploit this by including a malicious token alongside legitimate reward tokens:

1. Create an intent with legitimate rewards (e.g., 100 USDC) plus a seemingly worthless token (e.g., 1 XYZ).
2. The malicious XYZ token is programmed to revert on transfers.
3. A solver fills the intent, focusing on the USDC reward value.
4. When claiming rewards, the transfer of XYZ reverts, preventing access to the USDC.
5. After the deadline passes, the intent creator can reclaim all rewards.

This vulnerability can be used to create honeypot intents that appear profitable but are uncollectible.

- Solvers may not thoroughly validate every reward token.
- The malicious token could initially appear legitimate but be upgraded to revert.
- The economic incentive is significant as it allows reclaiming valuable tokens.

Impact Explanation: The impact is high. This vulnerability allows intent creators to effectively steal from solvers by preventing them from claiming legitimate rewards after they've fulfilled the intent.

Likelihood Explanation: The likelihood is medium. While it requires malicious intent from the intent creator, the attack is relatively simple to execute by creating a malicious token and including it alongside legitimate rewards. The potential for profit makes this an attractive attack vector.

Recommendation: Consider individual reward token transfers to fail using low level calls.

Eco: Fixed in commit 73b7652.

Cantina Managed: Verified that commit 73b7652 fixes the issue by handling transfer calls as recommended.

COMMENTS:

- ***Nishaad* (client)**: should suffice to just transfer via low-level call, right?

3.2.3 Missing enforcement of `route.destination` check may allow fillers to solve intents on other supported chains

Severity: High Risk

Context: Eco7683DestinationSettler.sol#L46-L107, Inbox.sol#L380-L446

Summary: Missing enforcement of `route.destination == block.chainId` check may allow fillers to solve intents on other supported chains leading to unexpected outcomes for intent creator.

Finding Description: `route.destination` encodes the target/destination chain ID where the intent calls should be executed by fillers. However, neither `Eco7683DestinationSettler.fill()` nor `Inbox._fulfill()` enforce a `route.destination == block.chainId` check, which allows fillers to accidentally/intentionally fill intents on other supported chains different from `route.destination`. This

may succeed if the intent's `route.calls` exist at the same addresses on such supported chains. Fillers can prove their intent filling nevertheless and claim their rewards.

Illustrative Scenario:

1. Intent creator Carol publishes an intent on Base to transfer 1 ether to Dave's SAFE Multisig account D on Arbitrum.
2. Intent filler Frank mistakenly fills Carol's intent on OP Mainnet instead of Arbitrum.
3. Dave did not have his SAFE Multisig at the same address D on OP Mainnet.
4. Dave does not receive his 1 ether on Arbitrum. This leads to loss of funds for Carol and Dave.
5. Frank receives his reward of 1 ether reward on Arbitrum. If Frank controls/creates address D on OP Mainnet, they would effectively profit from this incorrect filling.

As communicated by the client, this scenario could be pretty severe for native gas token transfer intent — e.g., someone requests 1 ETH on OP Mainnet to be sent to Base but exploiter fills on Polygon where native gas token POL is orders of magnitude cheaper than 1 ETH currently.

Impact Explanation: High, because the intent creator expected their intent to be filled on `route.destination` but it actually got filled on a different chain, which leads to unexpected outcome at the least and potential loss of funds in the worst case.

Likelihood Explanation: High for native gas token, because a malicious filler gets to profit orders of magnitude higher depending on the differential pricing of the native token between origin and destination chains.

Low for ERC20 tokens, because it depends on the existence of `route.calls` at the same addresses on the intent filled chain as `route.destination`. For example, DAI token is deployed at the same address on Arbitrum and OP Mainnet, but not on BASE. Also, it is not clear that fillers may always have an incentive for filling it on an incorrect chain.

Therefore, Medium overall.

Recommendation: Consider enforcing a `route.destination == block.chainId` in `Inbox._fulfill()` so that it is enforced on all intent fulfillment flows.

Eco: Fixed in commit 2e7bc2f.

Cantina Managed: Reviewed that commit 2e7bc2f fixes this issue with the below logic:

```
function _fulfill(
     Route memory _route,
     bytes32 _rewardHash,
     address _claimant,
     bytes32 _expectedHash
 ) internal returns (bytes[] memory) {
     if (_route.destination != block.chainid) {
         revert WrongChain(_route.destination);
     }
 // ...
```

3.3 Medium Risk

3.3.1 Missing enforcement of `openDeadline` check breaks ERC-7683 requirement and user expectations

Severity: Medium Risk

Context: Eco7683OriginSettler.sol#L88-L122, Eco7683OriginSettler.sol#L211, Eco7683OriginSettler.sol#L310

Summary: Missing enforcement of `openDeadline` check for gasless intents breaks ERC-7683 requirement.

Finding Description: ERC-7683 specifies requirements for gasless cross-chain intent orders on behalf of a user, which are created via `openFor()` and `resolveFor()` functions. One of the requirements is the enforcement of `openDeadline`, which is the timestamp by which the order must be opened by the filler on behalf of the user. This is part of `struct GaslessCrossChainOrder` and allows users to specify time-sensitive intents:

```
/// @dev The timestamp by which the order must be opened
uint32 openDeadline;
```

However, `openDeadline` check is not enforced against `block.timestamp` in `openFor()`, which allows fillers to open such intents at any time even beyond the deadline specified.

Impact Explanation: Medium, because users may have their intents opened by fillers beyond their specified `openDeadline`, which will lead to time-sensitive intents (e.g. cross-chain arbitrage opportunities in volatile market conditions) getting published and filled breaking user expectations and a ERC-7683 requirement.

Likelihood Explanation: Medium, because this only applies to gasless intents and it's not clear if how strict of a ERC-7683 requirement this is given that it is still in draft-mode.

Recommendation: Consider enforcing `openDeadline` check in `openFor()` to be defensively compliant with any ERC-7683 requirement.

Eco: Fixed in commit 2e7bc2f.

Cantina Managed: Reviewed that commit 2e7bc2f fixes this issue with the below logic:

```
function openFor(
    GaslessCrossChainOrder calldata _order,
    bytes calldata _signature,
    bytes calldata _originFillerData
) external payable override {
    if (block.timestamp > _order.openDeadline) {
        revert OpenDeadlinePassed();
    }
    // ...
```

and the use of `_order.openDeadline` in related logic.

3.3.2 Missing enforcement of `fillDeadline` check breaks ERC-7683 requirement and may lead to loss of filler rewards

Severity: Medium Risk

Context: Eco7683DestinationSettler.sol#L46-L107, ERC7683.sol#L14-L29, ERC7683.sol#L33-L43, ERC7683.sol#L53-L73

Summary: Missing enforcement of `fillDeadline` check in `Eco7683DestinationSettler.fill()` breaks ERC-7683 requirement and may lead to loss of filler rewards.

Finding Description: ERC-7683 specifies requirements for cross-chain intent orders on behalf of a user, which are filled via `fill()` function. One of the requirements is the enforcement of `fillDeadline`, which is the timestamp by which the order must be filled by the filler on the destination chain. This is part of all the ERC-7683 structs and allows users to specify time-sensitive intents:

```
* @param fillDeadline The timestamp by which the order must be filled on the destination chain
```

However, `fillDeadline` check is not enforced against `block.timestamp` in `Eco7683DestinationSettler.fill()`, which allows fillers to accidentally fill such intents at any time even beyond the `fillDeadline` specified. But such fillers cannot withdraw their rewards because this check is enforced during `withdrawRewards()` within `IntentVault.constructor()` when rewards for expired intents are refunded to intent creator.

Impact Explanation: High, because this missing check breaks ERC-7683 requirement and may lead to loss of filler rewards.

Likelihood Explanation: Low, because intent fillers are expected to perform their own due-diligence checks on intents before attempting to fill them and so this can only happen accidentally with fillers who do not self-validate intents correctly.

Recommendation: Consider adding a defensive `fillDeadline` check such as `block.timestamp <= order.fillDeadline` in `Eco7683DestinationSettler.fill()` to meet ERC-7683 requirement and prevent accidental loss of filler rewards.

Eco: Fixed in commit 2e7bc2f.

Cantina Managed: Reviewed that commit 2e7bc2f fixes this issue with the below logic:

```
function fill(
    bytes32 _orderId,
    bytes calldata _originData,
    bytes calldata _fillerData
) external payable {
    Intent memory intent = abi.decode(_originData, (Intent));
    if (block.timestamp > intent.reward.deadline) {
        revert FillDeadlinePassed();
    }
// ...
```

3.3.3 ERC-7683 intents can never be filled because of incorrectly encoded `originData`

Severity: Medium Risk

Context: Eco7683DestinationSettler.sol#L55-L56, Eco7683OriginSettler.sol#L185-L193, Eco7683OriginSettler.sol#L279-L285

Summary: ERC-7683 intents can never be filled in the protocol because incorrect `originData` is encoded in `FillInstruction.originData`.

Finding Description: Eco adds ERC-7683 compliant wrappers to allow ERC-7683 aware fillers to interoperate with the protocol. These wrappers are implemented in `Eco7683OriginSettler.sol` on the origin/source chains and `Eco7683DestinationSettler.sol` on the destination chains. `Eco7683OriginSettler` implements `open()`, `openFor()`, `resolve()` and `resolveFor()` functions and `Eco7683DestinationSettler` implements `fill()` function per ERC-7683.

However, while resolving an `OnchainCrossChainOrder` to a `ResolvedCrossChainOrder` in `resolve()` and `resolveFor()`, `onchainCrosschainOrderData.route.calls[j]` is encoded instead of `onchainCrosschainOrderData` for `FillInstructionused.originData` that is expected to be "The data generated on the origin chain needed by the destinationSettler to process the fill." `Eco7683DestinationSettler.fill()` attempts to decode this as `OnchainCrosschainOrderData`, which will be incorrect.

Impact Explanation: Medium, because fillers cannot use the ERC-7683 wrappers and instead have to rely on Eco's native intent logic.

Likelihood Explanation: Medium, because an ERC-7683 aware filler picking up such an intent will call the corresponding `Eco7683DestinationSettler.fill()`, which should always revert because it attempts to decode `onchainCrosschainOrderData.route.calls[j]` as `OnchainCrosschainOrderData`. An incentivized sophisticated filler however could re-encode correctly before calling `fill()`.

Recommendation: Consider encoding `onchainCrosschainOrderData` for `struct FillInstruction.originData` in `Eco7683OriginSettler.resolve()` and `Eco7683OriginSettler.resolveFor()` so that it is decoded correctly in `Eco7683DestinationSettler.fill()`.

Eco: Fixed in commit 2e7bc2f.

Cantina Managed: Reviewed that commit 2e7bc2f fixes this issue by encoding/decoding `intent` with the below logic:

```
fillInstructions[0] = FillInstruction(
    uint64(onchainCrosschainOrderData.route.destination),
    bytes32(uint256(uint160(onchainCrosschainOrderData.route.inbox))),
    abi.encode(intent)
);
```

```
fillInstructions[0] = FillInstruction(
    uint64(gaslessCrosschainOrderData.destination),
    bytes32(uint256(uint160(gaslessCrosschainOrderData.inbox))),
    abi.encode(intent)
);
```

```
function fill(
    bytes32 _orderId,
    bytes calldata _originData,
    bytes calldata _fillerData
) external payable {
    Intent memory intent = abi.decode(_originData, (Intent));
    // ...
```

3.3.4 Intents can be published multiple times due to default claim state

Severity: Medium Risk

Context: IntentSource.sol#L204-L206

Summary: The IntentSource contract allows the same intent to be published multiple times due to the missing claim state initialization. This occurs because the default value for claim status (0) is the same as the `Initiated` status, making it impossible to distinguish between uninitialized and newly initiated intents.

Description: In the IntentSource contract's `publishIntent()` function, the check to prevent duplicate intents relies on comparing the claim status against `ClaimStatus.Initiated`:

```
function publishIntent(
    Intent calldata intent,
    bool fund
) external payable returns (bytes32 intentHash) {
    // ...
    (intentHash, routeHash, ) = getIntentHash(intent);

    if (claims[intentHash].status != uint8(ClaimStatus.Initiated)) {
        revert IntentAlreadyExists(intentHash);
    }

    emit IntentCreated(
        intentHash,
        route.salt,
        route.source,
        route.destination,
        route.inbox,
        route.tokens,
        route.calls,
        reward.creator,
        reward.prover,
        reward.deadline,
        reward.nativeValue,
        reward.tokens
    );
    // ...
}
```

An issue arises because default values in Solidity are initialized to 0, which is the same value as the `Initiated` status. This means `claims[intentHash].status` returns 0 for both uninitialized and initiated intents. As a result, intents can be published multiple times before being claimed or refunded, potentially leading to duplicate funding or causing confusion for fillers monitoring intent creation events.

Impact Explanation: The impact is low. While this doesn't directly lead to loss of funds, it can cause operational issues for fillers relying on intent creation events and lead to accidental duplicate funding if users don't track their own intent publications.

Likelihood Explanation: The likelihood is medium. This issue manifests by default in the normal operation of the protocol. Any intent can be republished until it reaches a terminal state (Claimed or Refunded). This could happen accidentally through UI errors, transaction retries, or user confusion about the publishing status of their intents.

Recommendation: Add an `Uninitialized` state (0) to the claim status enum and make `Initiated` a non-zero value. Then modify the `publishIntent()` function to properly set the initial state:

```
  enum ClaimStatus {
+     Uninitialized,
      Initiated,
      Claimed,
      Refunded
  }

  function publishIntent(
      Intent calldata intent,
      bool fund
  ) external payable returns (bytes32 intentHash) {
      // ...
      (intentHash, routeHash, ) = getIntentHash(intent);

-     if (claims[intentHash].status != uint8(ClaimStatus.Initiated)) {
+     if (claims[intentHash].status != uint8(ClaimStatus.Uninitialized)) {
          revert IntentAlreadyExists(intentHash);
      }

+     claims[intentHash].status = uint8(ClaimStatus.Initiated);

      emit IntentCreated(
          intentHash,
          route.salt,
          route.source,
          route.destination,
          route.inbox,
          route.tokens,
          route.calls,
          reward.creator,
          reward.prover,
          reward.deadline,
          reward.nativeValue,
          reward.tokens
      );
      // ...
  }
```

This change ensures that new intents start in an `Uninitialized` state, publishing explicitly sets them to `Initiated`, and duplicate publishing attempts will fail. The result is a more robust intent lifecycle tracking system that prevents unintended republishing of intents.

Eco: We'll make the change on the 7683OriginSettler, but outside of that there is no case where somebody would double fund an intent. creating a duplicate intent is not really that big of a problem (and since we're not storing state it will be difficult to prevent it, since publishing an intent is not even a required step) and it wont be possible for a duplicate intent to be fulfilled, so we dont feel a need to change this. The state names are confusing though, we;ll look into those.

Cantina Managed: Acknowledged.

3.3.5 Missing nonce tracking in EIP-7683 integration allows for duplicate intent openings

Severity: Medium Risk

Context: Eco7683OriginSettler.sol#L319-L339

Summary: The Eco7683OriginSettler contract's `openFor()` function fails to track and validate nonces, allowing multiple openings of the same intent. This can lead to unauthorized token transfers from users who have granted allowances to the intent vault.

Description: The Eco7683OriginSettler contract is designed to integrate with the EIP-7683 standard, which includes nonce functionality to prevent replay attacks. However, the implementation is stateless and does not track nonces.

The `nonce` parameter from EIP-7683 is only used as salt in the Route struct, and IntentSource allows publishing the same intent multiple times. This means a valid signature can be reused to open the same intent repeatedly, allowing a malicious relayer to obtain a valid user signature and use it to call `openFor()` multiple times, transferring additional tokens from the user's account if they have granted sufficient allowance to the vault.

Impact Explanation: The impact is medium. While users can eventually recover excess transferred tokens as the intent creator, this vulnerability creates significant usability issues in the protocol. Any excess token transfers would require manual recovery by the user, creating friction and potential confusion about intent status and token balances. This behavior also represents a meaningful deviation from EIP-7683's security model, which relies on nonce tracking to prevent replay attacks.

Likelihood Explanation: The likelihood is low since exploitation requires specific conditions and offers limited benefits to attackers. Users must have granted excess token allowances to make the attack worthwhile, and malicious relayers gain no direct financial benefit from the exploitation. The time and effort required to execute this attack, combined with the ability of users to revoke allowances, make this an unlikely target for malicious actors.

Recommendation: Implement proper nonce tracking in the Eco7683OriginSettler contract:

```
+ mapping(address => uint256) public nonces;

  function _verifyOpenFor(
      GaslessCrossChainOrder calldata _order,
      bytes calldata _signature
  ) internal view returns (bool) {
      if (_order.originSettler != address(this)) {
          return false;
      }
+     if (_order.nonce != nonces[_order.user]++) {
+         revert InvalidNonce();
+     }

      // ... rest of the function ...
  }
```

Alternatively, by disallowing publishing intents multiple times in IntentSource, the `nonce` parameter (used as salt when publishing events) could prevent duplicate intent openings.

Eco: Fixed in PR 173.

Cantina Managed: Partially fixed by conditionally transferring tokens when the intent is not yet funded. `_openEcoIntent` in `Eco7683OriginSettler` now checks `if (!IntentSource(INTENT_SOURCE).isIntentFunded(_intent))` to make sure that intents are not funded twice. This still allows for opening intents multiple times, however, it reduces the impact of this particular issue.

3.3.6 Fund intent allows creation with invalid or incorrect route parameters

Severity: Medium Risk

Context: IntentSource.sol#L120-L128

Summary: The `fundIntent()` function in the IntentSource contract accepts a `routeHash` parameter without validating the underlying route components. This allows intents to be funded with routes that contain invalid parameters or are associated with incorrect chain IDs. Additionally, the `publishIntent()` function allows funding of intents (`fund = true`) even when `route.source` does not match `block.chainid`.

Description: The `fundIntent()` function accepts a pre-computed `routeHash` without requiring or validating the actual route parameters:

```
function fundIntent(
    bytes32 routeHash,
    Reward calldata reward,
    address fundingAddress,
    Call[] calldata permitCalls,
    address recoverToken
) external payable {
    bytes32 rewardHash = keccak256(abi.encode(reward));
    bytes32 intentHash = keccak256(abi.encodePacked(routeHash, rewardHash));

    address vault = _getIntentVaultAddress(intentHash, routeHash, reward);
    // ...
}
```

Similarly, the `publishIntent()` function allows intents to be funded even when they reference a different source chain:

```
function publishIntent(
    Intent calldata intent,
    bool fund
) external payable returns (bytes32 intentHash) {
    // No validation of intent.route.source
    if (fund && !_isIntentFunded(intent, vault)) {
        // Transfers tokens even if route.source != block.chainid
    }
}
```

This creates two issues:

1. The `routeHash` could be computed from invalid or incorrect route parameters.
2. Intents can be funded on chains that don't match their source chain ID, which shouldn't be possible as funding sources must be on the source chain.
3. This could mislead solvers to fill intents without being able to claim their rewards.

Impact Explanation: The impact is high as it could lead to funds being locked in vaults until expiration that are associated with invalid routes or incorrect chain IDs, making it impossible for fillers to claim rewards.

Likelihood Explanation: The likelihood is medium as these issues could occur through both malicious intent and accidental misconfiguration, especially when teams are integrating with the protocol's cross-chain mechanics.

Recommendation: Modify both functions to validate route parameters and ensure funding only occurs on the correct chain:

```
  function fundIntent(
-     bytes32 routeHash,
+     Intent calldata intent,
      address fundingAddress,
      Call[] calldata permitCalls,
      address recoverToken
  ) external payable {
+     if (intent.route.source != block.chainid) {
+         revert InvalidSourceChain();
+     }

+     bytes32 routeHash = keccak256(abi.encode(intent.route));
      bytes32 rewardHash = keccak256(abi.encode(intent.reward));
      // ...
  }

  function publishIntent(
      Intent calldata intent,
      bool fund
) external payable returns (bytes32 intentHash) {
+     if (fund && intent.route.source != block.chainid) {
+         revert CannotFundFromNonSourceChain();
+     }

      // ... rest of function
  }
```

These changes ensure route parameters are valid and that intents can only be funded on their source chain.

Eco: Fixed in commit 2e7bc2f.

Cantina Managed: Intents can still be published and funded on an incorrect chain id which could be misleading for fillers:

- `isIntentFunded()` checks `if (intent.route.source != block.chainid) return false;`.
- `publishAndFund()` and `publishAndFundFor()` call `_validateSourceChain(intent.route.source, intentHash)`; which checks:

```
  if (sourceChain != block.chainid) {
      revert WrongSourceChain(intentHash);
  }
```

- However, `publish()`, `fund()` and `fundFor()` are missing this check.

3.4 Low Risk

3.4.1 Opening ERC-7683 gasless cross-chain orders on behalf of users may lead to loss of native tokens for fillers

Severity: Low Risk

Context: Eco7683OriginSettler.sol#L354-L365

Summary: Opening ERC-7683 gasless cross-chain orders on behalf of users via `openFor()` may lead to loss of native tokens for fillers.

Finding Description: ERC-7683 allows opening of gasless cross-chain orders by fillers on behalf of users via `openFor()`. While `reward.tokens` are transferred from the user to the intent vault, `reward.nativeValue` is incorrectly transferred from `msg.sender` i.e. the filler to the intent vault.

Impact Explanation: Medium, because this may lead to loss of native tokens for fillers who call `openFor()`. These reward native tokens are claimed by any filler that later solves the intent on the destination chain, which may be different from the one who opened the intent on the source chain.

Likelihood Explanation: Low, because:

1. Intent creators may have an offchain native token refund mechanism for such fillers.
2. Intent fillers are expected to be sophisticated (e.g. would simulate such transactions before executing) and would not open intents if it leads to loss of funds for them.

Recommendation: Consider or clarify in documentation how fillers on source chain calling `openFor()` are compensated for their native reward token contribution.

Eco: Fixed in commit 2e7bc2f.

Cantina Managed: Reviewed that commit 2e7bc2f acknowledges this issue by noting for `openFor()` that:

```
* @notice This method is made payable in the event that the caller of this method (a solver) wants to open
* an intent that has native token as a reward. In this case, the solver would need to send the native
* token as part of the transaction. How the intent's creator pays the solver is not covered by this method.
```

3.4.2 Missing checks for overfunding of intents will lock up creator funds until rewards withdrawal or intent expiration

Severity: Low Risk

Context: Eco7683OriginSettler.sol#L348-L375, IntentSource.sol#L120-L182, IntentSource.sol#L225-L253, IntentSource.sol#L366-L383

Summary: Missing checks for intents will lock up creator funds until rewards withdrawal or intent expiration.

Finding Description: If an intent is funded with native or ERC20 tokens beyond amounts specified in `intent.reward` then those excess funds are unnecessarily locked in the vault until rewards withdrawal or intent expiration.

While `fundIntent()` funds the vault only with the deficit token amounts, there are other flows which do not perform deficit checks and fund in entirety leading to overfunding:

1. `publishIntent()` attempts to prevent overfunding by checking with `_isIntentFunded()` before funding which returns false for partially funded vaults and in such scenarios it proceeds to fund the vault again in entirety.
2. `Eco7683OriginSettler.open()` and `Eco7683OriginSettler.openFor()` do not perform reward deficit check and fund the intent vault unconditionally.

Creators calling `publishIntent()` (on partial funded intents) or `Eco7683OriginSettler.open()` multiple times will accidentally overfund their intent vault. Fillers calling `Eco7683OriginSettler.openFor()` multiple times will do the same. Malicious fillers may grief by intentionally calling this multiple times.

Impact Explanation: Low, because while the intent creator does not lose funds, their excess funds are locked up until rewards withdrawal (after intent filling and proving) or intent expiration at which time they are refunded.

Likelihood Explanation: Low, because this requires overfunding intent vaults accidentally or for griefing reasons.

Recommendation: Consider performing a `_isIntentFunded()` check and funding only the deficit amounts along all intent funding flows to prevent overfunding.

Eco: Fixed in commit 2e7bc2f.

Cantina Managed: Reviewed that commit 2e7bc2f partially fixes this issue with the below logic:

```
function _openEcoIntent(
    Intent memory _intent,
    address _user
) internal returns (bytes32 intentHash) {
    if (!IntentSource(INTENT_SOURCE).isIntentFunded(_intent)) {
        address vault = IntentSource(INTENT_SOURCE).intentVaultAddress(
            _intent
        );
// ...
```

which performs the funded check but does not address the deficit amounts in any partially funded scenario.

While `publishAndFundFor()` and `fundFor()` fund only the deficit amounts in `Vault._fundIntent()` via `_fundIntentFor()`, `publishAndFund()` and `fund()` fund the vault with the entire reward token amount via `_fundIntent()` depending on `_validateInitialFundingState()`.

3.4.3 Missing validation of `orderDataType` and incorrect EIP-712 encoding may break ERC-7683 requirement

Severity: Low Risk

Context: EcoERC7683.sol#L21-L47, EcoERC7683.sol#L49-L55

Description: ERC-7683 specifies that: "A compliant cross-chain order type MUST be ABI decodable into either `GaslessCrossChainOrder` or `OnchainCrossChainOrder` type. Both these structs include a `bytes32 orderDataType` which is meant to be: "Type identifier for the order data. This is an EIP-712 typehash.".

However, `OnchainCrossChainOrder.orderDataType` and `GaslessCrossChainOrder.orderDataType` are missing validation against `ONCHAIN_CROSSCHAIN_ORDER_DATA_TYPEHASH` and `GASLESS_CROSSCHAIN_ORDER_DATA_TYPEHASH`.

Moreover, these typehashes are incorrectly encoded:

1. `ONCHAIN_CROSSCHAIN_ORDER_DATA_TYPEHASH` encodes an incorrect name `EcoOnchainGaslessCrosschainOrderData` instead of `EcoOnchainCrosschainOrderData`.
2. Struct names should match those in typehashes.
3. Route is missing `bytes32 salt`.
4. Route is missing `TokenAmount[] tokens`.
5. `EcoGaslessCrosschainOrderData` is missing `TokenAmount[] routeTokens`.

Recommendation: Consider including validation of `orderDataType` against their respective typehashes after fixing the typehashes.

Eco: Fixed in commit 2e7bc2f.

Cantina Managed: Reviewed that commit 2e7bc2f fixes this issue with the below logic:

```
//EIP712 typehashes
bytes32 constant ONCHAIN_CROSSCHAIN_ORDER_DATA_TYPEHASH = keccak256(
    "OnchainCrosschainOrderData(Route route,address creator,address prover,uint256 nativeValue,TokenAmount[]
    ↪  rewardTokens)Route(bytes32 salt,uint256 source,uint256 destination,address inbox,TokenAmount[]
    ↪  tokens,Call[] calls)TokenAmount(address token,uint256 amount)Call(address target,bytes data,uint256
    ↪  value)"
);
bytes32 constant GASLESS_CROSSCHAIN_ORDER_DATA_TYPEHASH = keccak256(
    "GaslessCrosschainOrderData(uint256 destination,address inbox,TokenAmount[] routeTokens,Call[]
    ↪  calls,address prover,uint256 nativeValue,TokenAmount[] rewardTokens)TokenAmount(address token,uint256
    ↪  amount)Call(address target,bytes data,uint256 value)"
);
```

```
function open(
    OnchainCrossChainOrder calldata _order
) external payable override {
    if (_order.orderDataType != ONCHAIN_CROSSCHAIN_ORDER_DATA_TYPEHASH) {
        revert TypeSignatureMismatch();
    }
// ...
```

```
function openFor(
    GaslessCrossChainOrder calldata _order,
    bytes calldata _signature,
    bytes calldata _originFillerData
) external payable override {
    // ...
    if (_order.orderDataType != GASLESS_CROSSCHAIN_ORDER_DATA_TYPEHASH) {
        revert TypeSignatureMismatch();
    }
    // ...
```

3.4.4 Missing route and reward validations allows creating economically non-viable intents

Severity: Low Risk

Context: IntentSource.sol#L120-L254

Summary: The `IntentSource` contract lacks validation for route and reward parameters when creating intents. This allows the creation of intents that provide no economic value to fillers or could result in tokens being unintentionally locked in the inbox contract.

Description: Several important validations are missing when creating intents, such as:

1. No minimum reward requirement.
2. Route parameters can specify tokens without corresponding calls.

While fillers can validate intent viability off-chain, basic on-chain validations could prevent unintentional user errors like:

- Creating intents with no rewards that would never be filled.
- Transferring tokens to the inbox without corresponding calls to use them.
- Creating intents with mismatched token and call configurations.

Impact Explanation: The impact is medium. While the protocol remains functional and sophisticated fillers will avoid non-viable intents, the lack of validation could lead to user confusion or locked tokens in edge cases.

Likelihood Explanation: The likelihood is low as fillers will perform their own economic viability checks before executing intents, and most integrators will implement appropriate validation in their interfaces.

Recommendation: Add basic sanity checks for intent creation:

```
function validateIntent(Intent memory intent, bool fund) internal pure {
    // Validate chain configuration
    Route calldata route = intent.route;
    Reward calldata reward = intent.reward;

    // Basic route validation
    if (route.destination == 0) revert InvalidDestination();
    if (route.inbox == address(0)) revert InvalidInbox();
    if (route.source > type(uint32).max) revert InvalidSourceChain();
    if (fund && route.source != block.chainid) revert InvalidSourceChain();

    // Ensure there are calls to execute
    if (route.calls.length == 0) revert NoCalls();

    // Validate route tokens have corresponding calls
    if (route.tokens.length > route.calls.length) revert TooManyTokens();

    // Ensure there are rewards
    if (reward.tokens.length == 0 && reward.nativeValue == 0) {
        revert NoRewards();
    }

    // Validate reward amounts
    for (uint256 i = 0; i < reward.tokens.length; i++) {
        if (reward.tokens[i].amount == 0) revert InvalidRewardAmount();
    }

    // Validate timing
    if (reward.deadline <= block.timestamp) {
        revert InvalidDeadline();
    }
}
```

These validations provide basic safety guardrails while keeping in mind that fillers will make their own economic decisions about which intents to execute.

Eco: Solvers will have to perform their own profitability checks regardless, so theres no reason to have a minimum reward. solvers are also agnostic to what the user plans to do with the routeTokens that are transferred to the inbox, even if they do nothing the solver will factor into their profit check this net flow of tokens away from their wallet.

Cantina Managed: The idea is to implement these low-cost sanity checks for the intent creator's sake.

Eco: I think we are of the opinion that the solver is the best party to deal with maximum amounts of complexity. the more thin the onchain protections, the less gas for the average user and the more complex arbitrary stuff the system can do. So I'm ok leaving out the checks for now.

Cantina Managed: Acknowledged.

3.4.5 Permit call success check enables front-running DoS

Severity: Low Risk

Context: IntentSource.sol#L162-L166

Description: In the IntentSource contract's `fundIntent()` function, each permit call is required to succeed:

```
for (uint256 i = 0; i < callsLength; ++i) {
    Call calldata call = permitCalls[i];
    (bool success, ) = call.target.call(call.data);
    if (!success) {
        revert PermitCallFailed();
    }
}
```

This allows for a front-running attack where an attacker executes one of the permit calls directly with the token contract before the intent funding transaction. When the original transaction executes, it will revert because the already-executed permit call fails, preventing the intent from being funded.

Recommendation: Remove the success check for permit calls and let them fail silently:

```
  for (uint256 i = 0; i < callsLength; ++i) {
      Call calldata call = permitCalls[i];
-     (bool success, ) = call.target.call(call.data);
-     if (!success) {
-         revert PermitCallFailed();
-     }
+     // Permit calls can fail silently if already executed
+     call.target.call(call.data);
  }
```

This allows the funding transaction to succeed even if some permits have been front-run, requiring only that sufficient allowance exists for the token transfers.

Eco: Fixed in commit 2e7bc2f.

Cantina Managed: Reviewed that commit 2e7bc2f avoids this issue by removing permit calls.

3.4.6 Accidentally sent native currency is not refunded for intents

Severity: Low Risk

Context: Eco7683OriginSettler.sol#L354-L365, IntentSource.sol#L137-L155

Description: While `Eco7683OriginSettler._openEcoIntent()` refunds any `msg.value` sent in excess of `intent.reward.nativeValue`, it does so only if `intent.reward.nativeValue > 0`. An intent creator who accidentally sends native currency but with `intent.reward.nativeValue == 0` is not refunded their `msg.value`, which gets stuck in the `Eco7683OriginSettler` contract. The same issue is present in `IntentSource.fundIntent()`.

Recommendation: Consider moving the native currency refund logic outside the conditional check for `intent.reward.nativeValue > 0` in `Eco7683OriginSettler._openEcoIntent()` and similarly in `fundIntent()`.

Eco: Fixed in commit ce03451.

Cantina Managed: Reviewed that commit ce03451 fixes this issue with the below logic:

```
function _openEcoIntent(
    Intent memory _intent,
    address _user
) internal returns (bytes32 intentHash) {
    // ...
    payable(msg.sender).transfer(address(this).balance);
    // ...
```

```
function publishAndFund(
    Intent calldata intent
) external payable returns (bytes32 intentHash) {
    // ...
    _returnExcessEth(intentHash, address(this).balance);
}
```

```
function fund(
    bytes32 routeHash,
    Reward calldata reward,
    bool allowPartial
) external payable returns (bytes32 intentHash) {
    // ...
    _returnExcessEth(intentHash, address(this).balance);
}
```

3.4.7 Add contract existence checks when intent specifies calldata

Severity: Low Risk

Context: Inbox.sol#L432-L442

Description: The Inbox contract currently does not verify whether target addresses contain code when executing calls with calldata. This could lead to accidental cross-chain operations if a user accidentally specifies an incorrect address or an EOA (Externally Owned Account) instead of a contract address.

Recommendation: Add validation to check that target addresses contain code when calldata is present:

```
  for (uint256 i = 0; i < _route.calls.length; i++) {
      Call calldata call = _route.calls[i];
      if (call.target == mailbox) {
          revert CallToMailbox();
      }
+     if (call.data.length > 0 && call.target.code.length == 0) {
+         revert InvalidCallTarget(call.target);
+     }
      (bool success, bytes memory result) = call.target.call{value: call.value}(call.data);
     // ...
  }
```

This helps prevent users errors before cross-chain operations are filled.

Eco: Fixed in commit 2e7bc2f.

Cantina Managed: Reviewed that commit 2e7bc2f fixes this by adding the below check to `Inbox._fulfill()`:

```
// ...
if (call.target.code.length == 0 && call.data.length > 0) {
            // no code at this address
            revert CallToEOA(call.target);
}
// ...
```

3.4.8 Unsafe casting between `uint256` and `int256` values

Severity: Low Risk

Context: IntentFunder.sol#L60-L70, IntentSource.sol#L134-L140

Description: In the IntentSource contract, conversions between unsigned and signed integers are performed without checking for potential overflows. While this does not present a practical risk in the current implementation, it's a deviation from best practices for safe integer handling:

```
int256 vaultBalanceDeficit = int256(reward.nativeValue) -
    int256(vault.balance);

if (vaultBalanceDeficit > 0 && msg.value > 0) {
    uint256 nativeAmount = msg.value > uint256(vaultBalanceDeficit)
        ? uint256(vaultBalanceDeficit)
        : msg.value;
    // ...
}
```

Recommendation: Use OpenZeppelin's SafeCast library to handle integer type conversions:

```
+ import "@openzeppelin/contracts/utils/math/SafeCast.sol";

  contract IntentSource {
+     using SafeCast for uint256;
+     using SafeCast for int256;

      // ...

-     int256 vaultBalanceDeficit = int256(reward.nativeValue) -
-         int256(vault.balance);
+     int256 vaultBalanceDeficit = reward.nativeValue.toInt256() -
+         vault.balance.toInt256();

      if (vaultBalanceDeficit > 0 && msg.value > 0) {
-         uint256 nativeAmount = msg.value > uint256(vaultBalanceDeficit)
+         uint256 nativeAmount = msg.value > vaultBalanceDeficit.toUint256()
              ? uint256(vaultBalanceDeficit)
              : msg.value;
          // ...
      }
  }
```

Eco: Fixed in commit 2e7bc2f.

Cantina Managed: Reviewed that commit 2e7bc2f does not have the reported unsafe casting.

3.4.9 Prefer call over transfer for native token transfers

Severity: Low Risk

Context: Eco7683OriginSettler.sol#L359, IntentSource.sol#L142, IntentSource.sol#L231

Description: The protocol uses `.transfer()` for native token transfers in multiple locations. The `.transfer()` function forwards a fixed gas stipend of 2300 gas, which can cause transfers to fail if the receiving contract requires more gas for its receive/fallback functions. Although currently not an issue, it is seen as a best practice to user `.call()` whenever safe. This has historically been problematic as gas costs and requirements have changed through network upgrades.

Recommendation: Replace `.transfer()` with `.call()` for all native token transfers, where it is appropriate and safe.

Eco: Not worried about this since these are between our contracts so we dont have to worry about weird receives.

Cantina Managed: Acknowledged.

3.4.10 Restrict arbitrary calls in fund intent to permit operations only

Severity: Low Risk

Context: IntentSource.sol#L157-L162

Description: The IntentSource contract currently allows arbitrary calls to any address through the `permitCalls` parameter in `fundIntent()`:

```
function fundIntent(
    bytes32 routeHash,
    Reward calldata reward,
    address fundingAddress,
    Call[] calldata permitCalls,
    address recoverToken
) external payable {
    // ...
    for (uint256 i = 0; i < callsLength; ++i) {
        Call calldata call = permitCalls[i];
        (bool success, ) = call.target.call(call.data);
        if (!success) {
            revert PermitCallFailed();
        }
    }
    // ...
}
```

This design could lead to unauthorized token drains if a user accidentally grants permissions to the IntentSource contract, as any caller could execute arbitrary calls using those permissions.

Recommendation: Replace arbitrary calls with a strict interface that only allows permit operations:

```
function fundIntent(
    bytes32 routeHash,
    Reward calldata reward,
    address fundingAddress,
    PermitData[] calldata permits,
    address recoverToken
) external payable {
    // ...
    for (uint256 i = 0; i < permits.length; ++i) {
        PermitData calldata data = permits[i];
        try IERC20Permit(data.token).permit(
            data.owner,
            data.spender,
            data.value,
            data.deadline,
            data.v,
            data.r,
            data.s
        ) {} catch {};
    }
    // ...
}
```

This restricts operations to standard permit calls only, preventing potential misuse of accidentally granted permissions.

Eco: Fixed in commit 2e7bc2f.

Cantina Managed: Reviewed that commit 2e7bc2f does not have these arbitrary calls but instead funds reward tokens from a funding address and/or a `permitContact`.

3.5 Informational

3.5.1 Reward claims should be allowed until end of deadline

Severity: Informational

Context: IntentVault.sol#L39-L50

Description: In the IntentVault contract, reward claims are rejected if the current timestamp is equal to the deadline, since the check uses strictly less than (<) instead of less than or equal to (<=). This makes the effective deadline one second earlier than specified.

```
// Ensure intent has expired if there's no claimant
if (claimant == address(0) && block.timestamp < reward.deadline) {
    revert IntentNotExpired();
}

// Withdrawing to creator if intent is expired or already claimed/refunded
if (
    (claimant == address(0) && block.timestamp >= reward.deadline) ||
    state.status != uint8(IIntentSource.ClaimStatus.Initiated)
) {
    claimant = reward.creator;
}
```

Recommendation: Update the timestamp checks to be inclusive of the deadline by using less than or equal to:

```
  // Ensure intent has expired if there's no claimant
- if (claimant == address(0) && block.timestamp < reward.deadline) {
+ if (claimant == address(0) && block.timestamp <= reward.deadline) {
      revert IntentNotExpired();
  }

  // Withdrawing to creator if intent is expired or already claimed/refunded
  if (
-     (claimant == address(0) && block.timestamp >= reward.deadline) ||
+     (claimant == address(0) && block.timestamp > reward.deadline) ||
      state.status != uint8(IIntentSource.ClaimStatus.Initiated)
  ) {
      claimant = reward.creator;
  }
```

Eco: Fixed in commit 2e7bc2f.

Cantina Managed: Verified that commit 2e7bc2f allows claiming rewards up until the deadline.

3.5.2 Natspec can be improved for better code comprehension and UX

Severity: Informational

Context: Eco7683OriginSettler.sol#L49, Eco7683OriginSettler.sol#L81-L92, Inbox.sol#L38, IntentSource.sol#L83, IntentSource.sol#L309, IntentSource.sol#L391, IInbox.sol#L66

Description: Some aspects of Natspec are either incomplete or incorrect, which reduces code comprehension. Some illustrative examples are noted below:

1. "when intent solving is made public" → "when a batch of fulfilled intents is sent to Hyperlane mailbox with relayer support".
2. "Batch withdraws multiple intents with the same claimant" → "Batch withdraws multiple intents".
3. "The calculated vault address" → "The calculated funder address".
4. "intent Intent to calculate vault address for" → "intent Intent to calculate funder address for".
5. "opens an Eco intent directly on chaim" → "opens an Eco intent directly on chain".
6. `openFor()` Natspec is copied over from `open()` → Update appropriately.
7. "Privileged functions are designed to only allow one-time changes" → While this is true for `setMailbox()` and `makeSolvingPublic()`, it is not true for `changeSolverWhitelist()`, which allows toggling of whitelist status any number of times.

Recommendation: Consider improving the Natspec for better code comprehension and UX.

Eco: Fixed in commit 2e7bc2f.

Cantina Managed: Reviewed that commit 2e7bc2f fixes this issue with the required Natspec changes.

3.5.3 Protocol design recommendations

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description and Recommendations:

1. Improve Intent Vault & Intent Funder Action Clarity. Make actions in IntentVault more explicit by separating the reward distribution, creator refund, and token recovery flows into distinct conditional blocks. Currently, these actions are overlapping in the logic. Restructure the flow to make each path explicit:

```
contract IntentVault {
    constructor(bytes32 intentHash, Reward memory reward) {
        IIntentSource intentSource = IIntentSource(msg.sender);
        IIntentSource.ClaimState memory state = intentSource.getClaim(intentHash);
        address refundToken = intentSource.getRefundToken();

        if (refundToken != address(0)) {
            _handleRefundToken(refundToken, reward.creator);
        }

        if (state.claimant != address(0)) {
            _handleSuccessfulClaim(state.claimant, reward);
        } else if (block.timestamp >= reward.deadline ||
                state.status != uint8(IIntentSource.ClaimStatus.Initiated)) {
            _handleExpiredOrInvalidClaim(reward.creator, reward);
        } else {
            revert IntentNotExpired();
        }

        selfdestruct(payable(reward.creator));
    }
}
```

2. Restrict Intent Publishing to Source Chain. Consider removing the ability to publish intents on chains other than the funding source chain. While currently supported, this feature adds complexity without clear benefits, as intents can be published off-chain instead. Publishing on other chains still requires interaction with the source chain for funding, making the current flexibility potentially confusing and unnecessary.
3. Require Full Funding for Intent Publication. Enforce full funding as a requirement for intent publication. The current system allows partially funded or unfunded intents to exist, creating ambiguous states that complicate intent tracking. Making funding mandatory would provide a clear signal to off-chain services about intent readiness and simplify the system's state management.
4. Reconsider Separate Funding Mechanism. Evaluate the necessity of allowing intent funding without publication. The current separation between funding and publication creates additional complexity in state tracking and potentially confusing user experiences. Combining these operations would simplify the protocol's usage pattern.
5. Event Emission. Consolidate intent-related events to emit a single event when an intent is both published and funded. This would provide a clearer signal to off-chain services and reduce the complexity of intent state tracking. The current multiple-event system requires additional logic to determine when an intent becomes actionable.
6. Unify Vault and Funder Functionality. Combine the IntentVault and IntentFunder contracts as they perform similar token handling operations. This consolidation would reduce code duplication and simplify the system's architecture. The current separation increases complexity in the codebase.
7. Add Explicit Token Destination. Introduce a `address[] tokensTo` parameter in the Route struct to explicitly specify token destinations. Currently, token routing must be inferred from encoded calls, making the system less transparent and more prone to errors. This addition would make token flow more obvious and reduce the risk of tokens becoming trapped in the Inbox contract.

Eco: Fixed in commit 2e7bc2f.

Cantina Managed: Reviewed that commit 2e7bc2f addresses (1) and (6).

3.5.4 Missing sanity checks on solver whitelisting may cause unexpected behavior

Severity: Informational

Context: Inbox.sol#L43-L52, Inbox.sol#L350-L369

Description: Protocol uses a solver whitelist to restrict intent fulfillment to whitelisted solvers until solving is made public. Once it is made public (by Inbox owner), it cannot be restricted again. However, this logic is missing two sanity checks which may cause unexpected behavior:

1. Constructor sets `isSolvingPublic` and `solverWhitelist`. But the whitelist is only applicable if `isSolvingPublic` is being set to false. The setting of `solverWhitelist` can be conditioned with a `if`

`(!isSolvingPublic)` check.

2. `changeSolverWhitelist` allows owner to change the whitelist state of a particular solver. However, this is only relevant if `isSolvingPublic` is still false. The setting of `solverWhitelist` can be conditioned with a `if (!isSolvingPublic)` check and revert otherwise.

Recommendation: Consider enforcing the missing sanity checks illustrated above.

Eco: We'll skip this one since we're going to be pulling that functionality out soon anyway, and it doesnt introduce any problems.

Cantina Managed: Acknowledged.

3.5.5 Missing emit of `orderFilled` event in `Eco7683DestinationSettler` may affect offchain monitoring

Severity: Informational

Context: Eco7683DestinationSettler.sol#L25-L30, Eco7683DestinationSettler.sol#L85-L104

Description: `Eco7683DestinationSettler` declares an `event orderFilled(bytes32 _orderId, address _solver)` which is supposed to be "Emitted when an intent is fulfilled using Hyperlane instant proving". However, this event is never emitted which may affect offchain monitoring.

Recommendation: Consider emitting this event appropriately in `fill()` or removing this unused declaration if events from `fulfillHyperInstantWithRelayer()` are sufficient.

Eco: Fixed in commit 2e7bc2f.

Cantina Managed: Reviewed that commit 2e7bc2f fixes this issue by unconditionally emitting `OrderFilled(_orderId, msg.sender)` in `Eco7683DestinationSettler.fill()`.