Remora Audit Report

Prepared by Cyfrin

Version 1.0

Lead Auditors

0xStalin

November 20, 2025

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

4.1 Protocol summary

Remora is developing a Real-World Asset (RWA) protocol that makes institutional-grade real estate accessible to investors worldwide through compliant tokenization via fractional shares. This system allows investors to earn passive rental income through regular disbursements without the complexities and responsibilities associated with traditional property management.

The latest pull request introduces several significant enhancements and refactorings to improve payout mechanics, governance capabilities, and contract modularity:

- **Asynchronous Payout Distribution:** Implemented a new asynchronous payout mechanism. Governance can now create a payout intent to signal the upcoming distribution for the current dividend index. Subsequently, the actual funds can be distributed to ChildToken contracts in a decoupled manner, enabling more efficient and flexible batch processing.
- **Seize Dividends from Frozen Accounts:** Added functionality allowing governance to seize all dividends accrued by frozen accounts during the entire freeze period. This ensures that payouts corresponding to the frozen timeframe are redirected to the designated custodian, strengthening compliance and risk managemen controls.
- **Externalization of DocumentManager from the ChildToken contract**
- **Allowing special rewards for referals**

5 Audit Scope

The audit scope is at commit d6aeb237d2f6b54c924d59d242371ed59207b5be which represents an update from the last audit at commit 6365a9e970758605f973ce0319236805e4188986.

6 Executive Summary

Over the course of 3 days, the Cyfrin team conducted an audit on the Remora smart contracts provided by Remora. In this period, a total of 3 issues were found.

During the audit, we identified 1 Medium issue with the remainder being informational and gas optimizations.

The medium-severity vulnerability has been identified in the newly introduced governance feature that enables the seizure of dividend payouts accrued by frozen accounts throughout the entire duration of the freeze period.

- The issue could enable a form of double-spending: after an account's freeze is lifted, the previously frozen holder would be able to successfully claim the full set of dividends corresponding to the freeze period a second time, despite those funds having already been legitimately seized and redirected to the designated custodian

Summary

Project Name	Remora
Repository	final-audit-prep
Commit	bb1aa1ba8279...
Fix Commit	255f1487372b...
Audit Timeline	November 17th - November 19th 2025
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	1
Low Risk	0
Informational	1
Gas Optimizations	1
Total Issues	3

Summary of Findings

[M-1] Seizing payouts for frozen users can lead to double spending if the holder is unfrozen in subsequent distributions	Resolved
[I-1] Users can reset the status of their `firstPurchase` on the `referralData` when the `stablecoin` doesn't revert on transfers to `address(0)`	Resolved
[G-1] Unnecessary usage of `nonReentrant` modifier on `ReferralManager::completeFirstPurchase`	Resolved

7 Findings

7.1 Medium Risk

7.1.1 Seizing payouts for frozen users can lead to double spending if the holder is unfrozen in subsequent distributions

Description: The `ChildToken::seizeFrozenFunds` function is designed to perform the following operations:

- Permit the legitimate holder to claim all dividend distributions accrued prior to the imposition of the freeze.
- Transfer to the designated custodian all dividend distributions corresponding to the entire freeze period, up to and including the most recent distribution paid at the time of seizure execution.

A vulnerability exists whereby a frozen holder may subsequently claim dividends attributable to the freeze period despite those funds having already been seized and redirected to the custodian.

The issue manifests under the following sequence of events:

1. `ChildToken::seizeFrozenFunds` is invoked on a frozen account while one or more distributions have occurred during the freeze period. This correctly redirects the frozen-period dividends to the custodian and records the seizure snapshot in `holder.frozenIndex` and related accounting variables.
2. A new dividend distribution is created after the seizure has taken place.
3. While the account remains frozen, the holder (or anyone) invokes `DividendManager::payoutBalance`. Because the account is still frozen, the internal accounting variable `holder.lastPayoutIndexCalculated` is forcibly reset to `holder.frozenIndex`.
4. The account is subsequently unfrozen.
5. Post-unfreeze, the holder again calls payoutBalance. At this point:

- The account is no longer frozen.
- `holder.lastPayoutIndexCalculated` remains at the value previously forced during step 3 (`holder.frozenIndex`).
- The payout routine therefore processes and credits all distributions from `holder.frozenIndex` through the current latest distribution index. Consequently, the holder receives the entirety of the previously seized frozen-period dividends a second time, resulting in a double payment (once received by the custodian and then by the holder).

Impact: The frozen user, who has already had their payouts seized for the duration of the freeze period, can regain access to claim those payouts, effectively taking funds that are reserved to process the payouts of other holders.

Proof of Concept: Add the next PoC to the `DividendManager.t.sol` test file:

```
    function test_PoC_DoubleSpendingPayoutsOfFrozenHolder() public {
        // distribute payout, freeze holder, distribute more payouts while holder is frozen,
        ↪  seizeFrozen, distribute a payout, call to payoutBalance() to reset lastIndex to frozenIndex,
        ↪  then unfreeze holder, call payoutBalance() and get access to all payouts since the user was
        ↪  frozen!
        address user = domesticUsers[0];
        address custodian = domesticUsers[1];

        uint64 tokenToMint = 5;
        uint64 payoutAmount = 100e6;

        // mint and send user the tokens
        _mintAndTransferToUser(user, tokenToMint);

        // create distribution + verify
        _fundPayoutToPaymentSettler(payoutAmount);
        assertEq(d_childTokenProxy.payoutBalance(user), payoutAmount);
```

```
        assertEq(d_childTokenProxy.payoutBalance(user), payoutAmount);

        // freeze user + 5 payouts
        d_childTokenProxy.freezeHolder(user);
        _fundPayoutToPaymentSettler(payoutAmount);
        _fundPayoutToPaymentSettler(payoutAmount);
        _fundPayoutToPaymentSettler(payoutAmount);
        _fundPayoutToPaymentSettler(payoutAmount);
        _fundPayoutToPaymentSettler(payoutAmount);

        // seize frozen payouts from user, send frozen funds to custodian
        d_childTokenProxy.seizeFrozenFunds(user, custodian, false);
        // user receives the payouts owed prior to being frozen
        assertEq(stableCoin.balanceOf(user), payoutAmount);
        // custodian receives the payouts while the user was frozen
        assertEq(stableCoin.balanceOf(custodian), payoutAmount * 5);
        assertEq(d_childTokenProxy.payoutBalance(user), 0);
        assertEq(d_childTokenProxy.isHolderFrozen(user), true);

        // One more payout - Since the user was frozen, this is the 6th payout
        _fundPayoutToPaymentSettler(payoutAmount);

        assertEq(d_childTokenProxy.payoutBalance(user), 0);

        d_childTokenProxy.unFreezeHolder(user);

        //@audit-issue => Because `frozenIndex` was not updated, bug allows the user to claim the 6
        ↪  payouts since he was frozen regardless that 5 of those 6 payouts have already been paid out
        ↪  to the custodian via the `seizeFrozenFunds()`
        assertEq(d_childTokenProxy.payoutBalance(user), payoutAmount * 6);

    }
```

Recommended Mitigation: When freezing the user again, set the `holderStatus.frozenIndex` to the `$._currentPayoutIndex`.

Remora: Fixed in commit 2969545

Cyfrin: Verified. `holderStatus.frozenIndex` is set to `$._currentPayoutIndex` after the holder is frozen again.

7.2 Informational

7.2.1 Users can reset the status of their `firstPurchase` on the `referralData` when the `stablecoin` doesn't revert on transfers to `address(0)`

Description: Users can create a referral to get a discount by calling `ReferralManager::createReferral`. The user receives a discount, and the referrer gets a bonus when the user makes their first purchase.

The system intends to give users a discount only once, but there is an edge case when the stablecoin allows transfer to address(0). This allows calling `ReferralManager::createReferral` and setting the `referrer` as `address(0)`. This effectively bypasses the check to validate if the user has already set a referrer and proceeds to set their `referralData.isFirstPurchase` as true, granting the discount to the user on the next purchase. This allows users to:

1. Call `ReferralManager::createReferral` setting `referrer` as address(0)
2. Purchase a token
3. Call `ReferralManager::createReferral` again setting `referrer` as address(0)

Impact: Users can game the referral system to receive a discount on all their purchases by resetting the `firstPurchase` status to true.

Recommended Mitigation: When creating the referral, validate that the `referrer` address is not the address(0). Alternatively, acknowledge this issue and make sure the signers never generate a signature for the `referrer` set as address(0).

Remora: Fixed in commit 20eddec

Cyfrin: Verified.

7.3 Gas Optimization

7.3.1 Unnecessary usage of `nonReentrant` modifier on `ReferralManager::completeFirstPurchase`

Description: `ReferralManager::completeFirstPurchase` has in place the `nonReentrant` modifier from the `ReentrancyGuardTransientUpgradeable` library, but this function is not susceptible to reentrancy.

- The caller is restricted to be the `TokenBank` contract, and the only external call is to do a transfer of stablecoin to the referrer.

As long as the stablecoin is set correctly to a valid contract, there is no need to use the `nonReentrant` modifier.

Recommended Mitigation: `nonReentrant` modifier is not required. Remove the import of `ReentrancyGuardTransientUpgradeable` library.

Remora Fixed in commit 59a33a4

Cyfrin: Verified.