Withdrawable NFT Adapter Audit Report

Prepared by CodeHawks

Version 2.0

Lead Eagles

SBSecurity (Slavchew & blckhv)

Holydevotion

September 17, 2025

Contents

1 About CodeHawks

CodeHawks is a leading competitive smart contract audit marketplace working to protect DeFi, protocols, their users, and funds from smart contract exploits through private and public smart contract security auditing competitions, joined by industry-leading auditors world-wide.

2 Disclaimer

The CodeHawks 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

Withdrawal OFT Adapter is a project which enables permissioned withdrawal of funds for migration to other cross-chain vendors, help protocols using LayerZero for cross-chain communication avoid vendor lock-in. This project is a minimal package that can be used to help opt-out of the LayerZero OFT system at some point down the line. Even though this codebase has been audited, during an audit, you should include the code from this package in scope as well as there are many pitfalls with integrating a package like this and LayerZero, which is not designed for people to opt-out.

5 Audit Scope

The audit scope was limited to:

```
src/DisableableOFT.sol
src/DisableableOFTAdapter.sol
src/DisableableOFTCore.sol
src/WithdrawableOFTAdapter.sol
```

6 Executive Summary

Over the course of 2 days, the CodeHawks team conducted an audit on the Withdrawable NFT Adapter smart contracts provided by Cyfrin. In this period, a total of 2 issues were found.

The findings consist of 1 Medium and 1 Low.

Summary

Project Name	Withdrawable NFT Adapter
Repository	withdrawable-oft-adapter
Commit	b5f627e22c83...
Fix Commit	ddf81e9a9315...
Audit Timeline	Sept 12th - Sept 15th
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	1
Low Risk	1
Informational	0
Gas Optimizations	0
Total Issues	2

Summary of Findings

[M-1] Double Redemption via Emergency Withdraw + Failed Payload Replay	Resolved
[L-1] `DisableableOFTCore::quoteSend` should also indicate send is paused	Resolved

7 Findings

7.1 Medium Risk

7.1.1 Double Redemption via Emergency Withdraw + Failed Payload Replay

Description: When using the `WithdrawableOFTAdapter` as a locker and the contracts are paused, followed by an emergency withdrawal done for a user who still possesses the OFT tokens:

1. as a failed LZ payload
2. on the source chain as OFT representation

Later on, when the contracts are unpaused, he'll be able to redeem his OFT tokens, resulting in the destination tokens unlocked from the Adapter.

This will be the second time he receives tokens based on the same OFT token he had when the emergency withdrawal was done.

Impact: The tokens redeemed later on, when the contracts are unpaused, will be other users' tokens, meaning that he steals their liquidity.

Proof of Concept: Here's a textual example:

1. Alice bridges 1000 tokens to the destination chain
2. Contracts paused (OFT Receive disabled)
3. Emergency withdrawal returns Alice's 1000 tokens
4. Contracts unpaused
5. Bob bridges 1000 tokens to the destination chain
6. Alice replays the failed message, stealing Bob's 1000 tokens
7. **Result**: Alice has 2000 tokens, Bob has 0 tokens

Also, coded POC:

1. The test can be placed in the `WithdrawableOFTAdapter.t.sol`
2. `TestableWithdrawableOFTAdapter` must be extended:

```
contract TestableWithdrawableOFTAdapter is WithdrawableOFTAdapter {
    constructor(address _token, address _lzEndpoint, address _delegate, address _owner, uint256
    ↪   _emergencyWithdrawDelay)
        WithdrawableOFTAdapter(_token, _lzEndpoint, _delegate, _owner, _emergencyWithdrawDelay)
    {}

+   function testCredit(address _to, uint256 _amountLD, uint32 _srcEid) external returns (uint256) {
+       return _credit(_to, _amountLD, _srcEid);
+   }
}
```

3. Execute with `forge test --match-test test_DoubleRedemptionVulnerability -vv`

```
function test_DoubleRedemptionVulnerability() public {
    token.mint(bob, 1000 ether);
    vm.deal(bob, 10 ether);

    uint256 amount = 1000 ether;
    bytes memory options = OptionsBuilder.newOptions().addExecutorLzReceiveOption(200000, 0);

    SendParam memory aliceSendParam = SendParam({
        dstEid: EID_B,
        to: bytes32(uint256(uint160(alice))),
        amountLD: amount,
        minAmountLD: amount,
        extraOptions: options,
        composeMsg: "",
        oftCmd: ""
    });

    SendParam memory bobSendParam = SendParam({
        dstEid: EID_B,
        to: bytes32(uint256(uint160(bob))),
        amountLD: amount,
        minAmountLD: amount,
        extraOptions: options,
        composeMsg: "",
        oftCmd: ""
    });

    MessagingFee memory fee = adapter.quoteSend(aliceSendParam, false);

    vm.prank(alice);
    token.approve(address(adapter), amount);
    vm.prank(alice);
    adapter.send{value: fee.nativeFee}(aliceSendParam, fee, alice);

    vm.prank(owner);
    adapter.setOFTReceive(false);

    vm.prank(owner);
    adapter.initializeEmergencyWithdraw(alice, amount);
    vm.warp(block.timestamp + EMERGENCY_DELAY + 1);
    vm.prank(owner);
    adapter.emergencyWithdraw();

    vm.prank(owner);
    adapter.setOFTSend(true);

    vm.prank(bob);
    token.approve(address(adapter), amount);
    vm.prank(bob);
    adapter.send{value: fee.nativeFee}(bobSendParam, fee, bob);

    adapter.testCredit(alice, amount, EID_B);

    assertEq(token.balanceOf(alice), amount * 2);
    assertEq(token.balanceOf(bob), 0);
}
```

Recommended Mitigation:

1. When tokens are in a failed payload you can `clear` it.
2. When tokens are on the other chain and in the user, you can burn them externally first by having a `burnFrom` function.

Withdrawable OFT Adapter: Fixed in commit 2fd3736.

Cyfrin: Verified.

7.2 Low Risk

7.2.1 `DisableableOFTCore::quoteSend` should also indicate send is paused

Description: `DisableableOFTCore::quoteSend` should also check for `s_isOFTSendEnabled`:

```
function _quote(
    uint32 _dstEid,
    bytes memory _message,
    bytes memory _options,
    bool _payInLzToken
) internal view virtual returns (MessagingFee memory fee) {
    return
        endpoint.quote(
            MessagingParams(_dstEid, _getPeerOrRevert(_dstEid), _message, _options, _payInLzToken),
            address(this)
        );
}
```

Impact: Currently the `quoteSend` will return the fee, but the send operation may fail and the contracts integrating with it will not be notified in earlier.

Recommended Mitigation: Override `_quote` in `DisableableOFTCore`.

Withdrawable OFT Adapter: Fixed in commit 2fd3736.

Cyfrin: Verified.