# Security Audit Report for DStock OFT

**Date:** December 25, 2025 **Version:** 1.0

**Contact:** contact@blocksec.com

# Contents

## Report Manifest

| Item | Description |
|---|---|
| Client | DStock |
| Target | DStock OFT |

## Version History

| Version | Date | Description |
|---|---|---|
| 1.0 | December 25, 2025 | First release |

## Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-02-24 07:44:35+0000

# Chapter 1 Introduction

## 1.1 About Target Contracts

| Information | Description |
|---|---|
| Type | Smart Contract |
| Language | Solidity |
| Approach | Semi-automatic and manual verification |

The target of this audit is the code repository [1] of DStock OFT of DStock.

DStock OFT delivers a bridgeable token system. DStockOFTUpgradeable extends LayerZero's OFT with an upgrade-safe proxy design plus operational controls: granular roles for pausing the bridge, maintaining a blacklist, confiscating balances, and diverting intercepted credits to a treasury.

Note this audit only focuses on the smart contracts in the following directories/files:

- src/*

Other files are not within the scope of the audit. Additionally, all dependencies of the smart contracts within the audit scope are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report. Code prior to and including the baseline version (`Version 0`), where applicable, is outside the scope of this audit and assumes to be reliable and secure.

| Project | Version | Commit Hash |
|---|---|---|
| dstock-oft | Version 1 | c41e984a3f8a745527979e40c4400d565c9738c1 |
| | Version 2 | 963b10473e6e125c83fbf4bd7770620d3683911a |

## 1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does

[1] https://github.com/dstockofficial/dstock-oft

not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

## 1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

### 1.3.1 Security Issues

* Access control
* Permission management
* Whitelist and blacklist mechanisms
* Initialization consistency
* Improper use of the proxy system
* Reentrancy
* Denial of Service (DoS)
* Untrusted external call and control flow
* Exception handling
* Data handling and flow
* Events operation
* Error-prone randomness
* Oracle security
* Business logic correctness
* Semantic and functional consistency
* Emergency mechanism
* Economic and incentive impact

### 1.3.2 Additional Recommendation

* Gas optimization
* Code quality and style

**Note** *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

## 1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

**Table 1.1:** Vulnerability Severity Classification

| | | | |
|---|---|---|---|
| **Impact** | *High* | High | Medium |
| | *Low* | Medium | Low |
| | | *High* | *Low* |
| | | **Likelihood** | |

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following five categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Partially Fixed** The item has been confirmed and partially fixed by the client.
- **Fixed** The item has been confirmed and fixed by the client.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

# Chapter 2 Findings

In total, we have **one** recommendation and **one** note.

- Recommendation: 1
- Note: 1

| ID | Severity | Description | Category | Status |
|---|---|---|---|---|
| 1 | - | Lack of check in function `initialize()` | Recommendation | Fixed |
| 2 | - | Potential centralization risks | Note | - |

The details are provided in the following sections.

## 2.1 Recommendation

### 2.1.1 Lack of check in function `initialize()`

**Status** Fixed in `Version 2`

**Introduced by** `Version 1`

**Description** In the `DStockOFTUpgradeable` contract, the function `initialize()` is responsible for performing the initial setup. However, the function does not validate whether the `_admin` parameter is a non-zero address. Since `_admin` cannot be modified again after initialization, incorrectly setting it to the zero address would render privileged functions unusable.

```
function initialize(
    string memory _name,
    string memory _symbol,
    address _lzDelegate,
    address _admin,
    address _treasury
) external initializer {
    __OFT_init(_name, _symbol, _lzDelegate);

    // Compatibility: some LayerZero OFTUpgradeable variants do not initialize Ownable.
    if (owner() == address(0)) {
        _transferOwnership(_lzDelegate);
    }

    __AccessControl_init();
    __Pausable_init();

    _grantRole(DEFAULT_ADMIN_ROLE, _admin);
    _grantRole(PAUSER_ROLE, _admin);
    _grantRole(UNPAUSER_ROLE, _admin);
    _grantRole(BLACKLIST_MANAGER_ROLE, _admin);
    _grantRole(CONFISCATOR_ROLE, _admin);

    interceptCreditToTreasury = true;
    emit InterceptCreditModeChanged(true);

```

```
        address t = _treasury == address(0) ? _admin : _treasury;
        if (t == address(0)) revert InvalidAddress(t);
        treasury = t;
        emit TreasuryUpdated(address(0), t);
    }
```

**Listing 2.1:** src/DStockOFTUpgradeable.sol

**Suggestion** Add a check to ensure that the `_admin` is not zero address.

## 2.2 Note

### 2.2.1 Potential centralization risks

**Introduced by** `Version 1`

**Description** In this project, several privileged roles (e.g., `DEFAULT_ADMIN_ROLE`, `BLACKLIST_MANAGER_ROLE`, and `CONFISCATOR_ROLE`) have the authority to perform sensitive operations, which introduces potential centralization risks. For example, these roles can update the `treasury` address or add arbitrary users to the blacklist. If the private keys associated with any of these privileged accounts are compromised or maliciously exploited, it could pose significant risks to the protocol and its users.

BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS