BLOCKSEC

Security Audit
Report for ListaOFT

Date: June 19, 2024 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Chapter 1 Introduction **1**
1.1 About Target Contracts . . . 1
1.2 Disclaimer . . . 1
1.3 Procedure of Auditing . . . 1
1.3.1 Software Security . . . 2
1.3.2 DeFi Security . . . 2
1.3.3 NFT Security . . . 2
1.3.4 Additional Recommendation . . . 2
1.4 Security Model . . . 3

Chapter 2 Findings **4**
2.1 DeFi Security . . . 4
2.1.1 Incorrect check of time difference in isMoreThanACalendarDay() . . . 4
2.1.2 Incorrect check in function _setTransferLimitConfig() . . . 5
2.1.3 Lack of dust removal in function _checkAndUpdateTransferLimit() . . . 7
2.2 Additional Recommendation . . . 8
2.2.1 Lack of check of duplicated configuratione . . . 8
2.3 Note . . . 9
2.3.1 Pontential centralization risk . . . 9

Report Manifest

Item	Description
Client	Lista
Target	ListaOFT

Version History

Version	Date	Description
1.0	June 19, 2024	First release

Signature

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 of ListaOFT[1] of Lista.

In the ListaOFT, users can initiate cross-chain token transfers powered by LayerZero.

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.

Project	Version	Commit Hash
ListaOFT	`Version 1`	`15679966e3e34a978156e5405d64d7b6f2bfa799`
	`Version 2`	`451b78eb50e96e0c5bbbcdb6045a3ecf0b0a981b`

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 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.

[1] https://github.com/lista-dao/lista-token/tree/feature/oft

- **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 Software Security

* Reentrancy
* DoS
* Access control
* Data handling and data flow
* Exception handling
* Untrusted external call and control flow
* Initialization consistency
* Events operation
* Error-prone randomness
* Improper use of the proxy system

1.3.2 DeFi Security

* Semantic consistency
* Functionality consistency
* Permission management
* Business logic
* Token operation
* Emergency mechanism
* Oracle security
* Whitelist and blacklist
* Economic impact
* Batch transfer

1.3.3 NFT Security

* Duplicated item
* Verification of the token receiver
* Off-chain metadata security

1.3.4 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 \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

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 four 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.
- **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 find **three** potential issue. Besides, we also have **one** recommendation and **one** note.

- Low Risk: 3
- Recommendation: 1
- Note: 1

ID	Severity	Description	Category	Status
1	Low	Incorrect check of time difference in isMoreThanACalendarDay()	DeFi Security	Fixed
2	Low	Incorrect check in function _setTransferLimitConfig()	DeFi Security	Fixed
3	Low	Lack of dust removal in function _checkAndUpdateTransferLimit()	DeFi Security	Fixed
4	-	Lack of check of duplicated configuration	Recommendation	Confirmed
5	-	Potential centralization risk	Note	-

The details are provided in the following sections.

2.1 DeFi Security

2.1.1 Incorrect check of time difference in isMoreThanACalendarDay()

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Function `isMoreThanACalendarDay()` is designed to compare and verify if the difference of two timestamps exceeds one calendar day. However, the calculation is incorrect due to the precision loss. For example, if `timestampB` is 86401 and `timestampA` is 86399, then `dayB` would be 1 and `dayA` would be 0, and the function would return `true`, even though the actual time difference has not truly exceeded one day.

```
148   /**
149    * @notice compare two timestamp and check if the difference is more than a calendar day
150    * @param timestampA timestamp A
151    * @param timestampB timestamp B
152    * @return true if the difference is more than a calendar day
153    */
154   function isMoreThanACalendarDay(uint256 timestampA, uint256 timestampB) internal virtual pure
          returns (bool) {
155     uint256 secondsPerDay = 86400; // 60 * 60 * 24
156     uint256 dayA = timestampA / secondsPerDay;
157     uint256 dayB = timestampB / secondsPerDay;
158
159
160     return dayB - dayA >= 1;
```

```
161   }
```

Listing 2.1: TransferLimiter.sol

Impact The reset to the restrictions on transfers are not proceeding as designed.

Suggestion Subtract for the difference of timestamp first, then divide by `secondsPerDay` to get actual elapsed days.

2.1.2 Incorrect check in function _setTransferLimitConfig()

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `TransferLimiter.sol`, the function `_setTransferLimitConfig()` is used to configure the parameters necessary for cross-chain token transfers. The function `_checkAndUpdateTransferLimit()` evaluates cross-chain requests based on these parameters. However, the conditions on lines 64-65 are contradictory to those on lines 119-131. Specifically, while a user's single cross-chain transfer amount can be equal to `singleTransferUpperLimit`, the conditions on lines 64-65 only ensure that `maxDailyTransferAmount` and `dailyTransferAmountPerAddress` are greater than `singleTransferLowerLimit`. This discrepancy may prevent the checks on lines 124 and 129 from passing, resulting in a revert.

Additionally, the function `_setTransferLimitConfig()` lacks validations for `maxDailyTransferAmount` and `dailyTransferAmountPerAddress`.

```
59    function _setTransferLimitConfig(TransferLimit memory limit) internal virtual {
60        // validate transfer limit config
61        require(limit.dstEid > 0, "dstEid must be greater than 0");
62        require(limit.singleTransferUpperLimit > limit.singleTransferLowerLimit, "upper limit must
              be greater than lower limit");
63        require(limit.dailyTransferAttemptPerAddress > 0, "dailyTransferAttemptPerAddress must be
              greater than 0");
64        require(limit.maxDailyTransferAmount > limit.singleTransferLowerLimit, "
              maxDailyTransferAmount must be greater than singleTransferLowerLimit");
65        require(limit.dailyTransferAmountPerAddress > limit.singleTransferLowerLimit, "
              dailyTransferAmountPerAddress must be greater than singleTransferLowerLimit");
66        // assign limit to the mapping
67        transferLimitConfigs[limit.dstEid] = limit;
68        // emit event
69        emit TransferLimitChanged(
70          limit.dstEid,
71          limit.maxDailyTransferAmount,
72          limit.singleTransferUpperLimit,
73          limit.singleTransferLowerLimit,
74          limit.dailyTransferAmountPerAddress,
75          limit.dailyTransferAttemptPerAddress
76        );
77      }
```

Listing 2.2: TransferLimiter.sol

```
 97   function _checkAndUpdateTransferLimit(uint32 _dstEid, uint256 _amount, address _user) internal
          virtual {
 98  TransferLimit memory limit = transferLimitConfigs[_dstEid];
 99  // check if transfer limit is set
100  if (limit.dstEid == 0) {
101    revert TransferLimitNotSet();
102  }
103  // check if amount is greater than 0
104  if (_amount == 0) {
105    revert TransferLimitExceeded();
106  }
107
108
109  // reset global transfer limit if the last transfer is made more than a calendar day
110  if (isMoreThanACalendarDay(lastUpdatedTime[_dstEid], block.timestamp)) {
111    dailyTransferAmount[_dstEid] = 0;
112  }
113  // reset user transfer limit and attempt if the last transfer is made more than a calendar day
114  if (isMoreThanACalendarDay(lastUserUpdatedTime[_dstEid][_user], block.timestamp)) {
115    userDailyTransferAmount[_dstEid][_user] = 0;
116    userDailyAttempt[_dstEid][_user] = 0;
117  }
118
119
120  // check if the transfer amount exceeds the upper and lower limit
121  if (_amount > limit.singleTransferUpperLimit || _amount < limit.singleTransferLowerLimit) {
122    revert TransferLimitExceeded();
123  }
124
125
126  // check if the transfer amount exceeds the daily transfer amount limit
127  if (dailyTransferAmount[_dstEid] + _amount > limit.maxDailyTransferAmount) {
128    revert TransferLimitExceeded();
129  }
130
131
132  // check if the transfer amount exceeds the daily transfer amount limit per address
133  if (userDailyTransferAmount[_dstEid][_user] + _amount > limit.dailyTransferAmountPerAddress) {
134    revert TransferLimitExceeded();
135  }
136
137
138  // check if the user exceeds the daily transfer attempt limit
139  if (userDailyAttempt[_dstEid][_user] >= limit.dailyTransferAttemptPerAddress) {
140    revert TransferLimitExceeded();
141  }
142  // update global transfer limit and updated timestamp
143  dailyTransferAmount[_dstEid] += _amount;
144  lastUpdatedTime[_dstEid] = block.timestamp;
145  // update user transfer limit and updated timestamp
146  userDailyTransferAmount[_dstEid][_user] += _amount;
147  userDailyAttempt[_dstEid][_user] += 1;
```

```
148   lastUserUpdatedTime[_dstEid][_user] = block.timestamp;
149 }
```

Listing 2.3: TransferLimiter.sol

Impact Contradictory logic in the conditions may cause legitimate cross-chain requests by users to be reverted.

Suggestion Revise the conditions on lines 64-65 to use `singleTransferUpperLimit` instead of `singleTransferLowerLimit`. Additionally, add a validation to ensure that `maxDailyTransferAmount` is greater than `dailyTransferAmountPerAddress`.

2.1.3 Lack of dust removal in function _checkAndUpdateTransferLimit()

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the function `super._debit()`, the parameter `_amountLD` is processed for dust removal before being used as the actual value burn. However, in the function `_checkAndUpdateTransferLimit()`, the recorded value is `_amountLD` without dust removal, resulting in a value higher than the actual burn amount.

```
70   function _debit(
71   address _from,
72   uint256 _amountLD,
73   uint256 _minAmountLD,
74   uint32 _dstEid
75 ) internal virtual override whenNotPaused returns (uint256 amountSentLD, uint256
       amountReceivedLD) {
76   _checkAndUpdateTransferLimit(_dstEid, _amountLD, _from);
77   return super._debit(_from, _amountLD, _minAmountLD, _dstEid);
78 }
```

Listing 2.4: ListaOFT.sol

```
50   function _debit(
51   address _from,
52   uint256 _amountLD,
53   uint256 _minAmountLD,
54   uint32 _dstEid
55 ) internal virtual override whenNotPaused returns (uint256 amountSentLD, uint256
       amountReceivedLD) {
56   _checkAndUpdateTransferLimit(_dstEid, _amountLD, _from);
57   return super._debit(_from, _amountLD, _minAmountLD, _dstEid);
58 }
```

Listing 2.5: ListaOFTAdapter.sol

Impact The `_amountLD` recorded in the function `_checkAndUpdateTransferLimit()` is higher than the actual burn amount.

Suggestion Remove dust from `_amountLD` in the function `_checkAndUpdateTransferLimit()`.

2.2 Additional Recommendation

2.2.1 Lack of check of duplicated configuratione

Status Confirmed

Introduced by Version 1

Description Function `setTransferLimitConfigs()` allows the privileged owner account to set an array of transfer limits for different destination chains. However, if the same `dstEid` is configured multiple times in the array, the previous settings will be overwritten, leading to unexpected results.

```
55   /**
56    * @notice sets the transfer limit configurations
57    * @param limit TransferLimit
58    */
59   function _setTransferLimitConfig(TransferLimit memory limit) internal virtual {
60     // validate transfer limit config
61     require(limit.dstEid > 0, "dstEid must be greater than 0");
62     require(limit.singleTransferUpperLimit > limit.singleTransferLowerLimit, "upper limit must be
         greater than lower limit");
63     require(limit.dailyTransferAttemptPerAddress > 0, "dailyTransferAttemptPerAddress must be
        greater than 0");
64     require(limit.maxDailyTransferAmount > limit.singleTransferLowerLimit, "
        maxDailyTransferAmount must be greater than singleTransferLowerLimit");
65     require(limit.dailyTransferAmountPerAddress > limit.singleTransferLowerLimit, "
        dailyTransferAmountPerAddress must be greater than singleTransferLowerLimit");
66     // assign limit to the mapping
67     transferLimitConfigs[limit.dstEid] = limit;
68     // emit event
69     emit TransferLimitChanged(
70       limit.dstEid,
71       limit.maxDailyTransferAmount,
72       limit.singleTransferUpperLimit,
73       limit.singleTransferLowerLimit,
74       limit.dailyTransferAmountPerAddress,
75       limit.dailyTransferAttemptPerAddress
76     );
77   }
78
79
80   /**
81    * @notice sets multiple transfer limit configurations are once
82    * @param _transferLimitConfigs an array of TransferLimit
83    */
84   function _setTransferLimitConfigs(TransferLimit[] memory _transferLimitConfigs) internal
         virtual {
85     for (uint256 i = 0; i < _transferLimitConfigs.length; i++) {
86       TransferLimit memory limit = _transferLimitConfigs[i];
87       _setTransferLimitConfig(limit);
88     }
89   }
```

Listing 2.6: TransferLimiter.sol

Suggestion Add a check to ensure there are no duplicate `dstEids` in the array during the configuration.

Feedback from the Project The team is aware of this situation, and it is within expectations.

2.3 Note

2.3.1 Pontential centralization risk

Introduced by `Version 1`

Description In the protocol, various limit checks exist, and the contract owner can modify these limit checks via function `setTransferLimitConfigs()`. If the owner's private key is lost or maliciously exploited, it could lead to losses for the protocol.

Feedback from the Project The team promises to use a multisig wallet for holding a critical role in the protocol.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS