BLOCKSEC

Security Audit

Report for Lista Dao Contracts

Date: August 14, 2024 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Lista
Target	Lista Dao Contracts

Version History

Version	Date	Description
1.0	August 14, 2024	First release

Signature

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

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2024-08-14 03:54:07+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

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 Lista Dao Contracts[1] of Lista. Note that, we did **NOT** audit all the modules in the repository. The modules covered by this audit report include `lista-dao-contracts` folder contract only. Specifically, the files covered in this audit include:

```
1 FlashBuy.sol
```

Listing 1.1: Audit Scope for this Report

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
Lista Dao Contracts	`Version 1`	`3cb89728f9633d9d9392c4f822aba6beaf345b22`
	`Version 2`	`03f5422f28721f2d0e701d4508589b58c4427949`

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.

[1] https://github.com/lista-dao/lista-dao-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 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

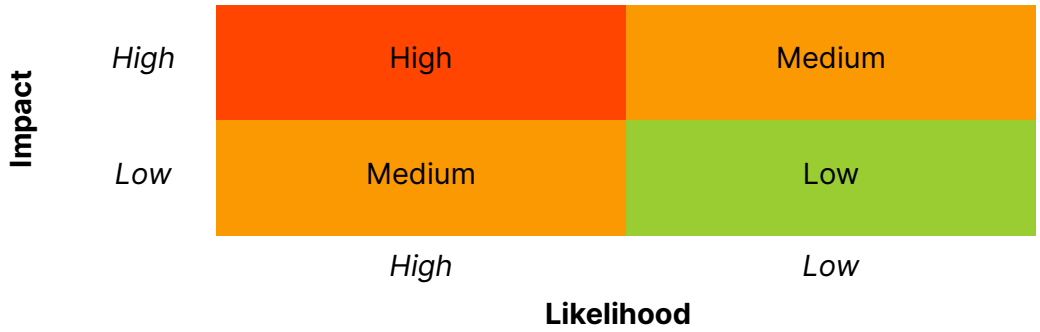
Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

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



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.

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

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

- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we find **three** potential issues, and **one** note as follows:

- High Risk: 3
- Note: 1

ID	Severity	Description	Category	Status
1	High	Sandwich attacks caused by incorrect calculation of `minOut`	Defi Security	Fixed
2	High	Potential DoS due to improper approve amount	Defi Security	Fixed
3	High	Failed `transferFrom()` due to contract's lack of approval for owner	Defi Security	Fixed
4	-	Interaction with out-of-scope contracts	Note	

The details are provided in the following sections.

2.1 DeFi Security

2.1.1 Sandwich attacks caused by incorrect calculation of `minOut`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Function `onFlashLoan()` is a callback function that utilizes tokens borrowed via `flash loan` to purchase collaterals from the `auction` and repay the borrowed tokens and `fees` by selling the acquired collaterals through a specified `DEX`. However, the calculation of `minOut`, which is used to control the swap slippage, is incorrect. Specifically, it calculates `minOut` based on the difference between the amount of the `flash loan` and `fees` to be repaid and the existing token quantity within the contract. In this scenario, assuming the contract itself holds a significant amount of tokens, it can result in a very small value for `minOut`, making it susceptible to `MEV` attacks during the swap.

```
128   function onFlashLoan(
129       address initiator,
130       address token,
131       uint256 amount,
132       uint256 fee,
133       bytes calldata data
134   ) external override returns (bytes32) {
135       require(
136           msg.sender == address(lender),
137           "FlashBorrower: Untrusted lender"
138       );
139       require(
140           initiator == address(this),
141           "FlashBorrower: Untrusted loan initiator"
```

```
142         );
143
144
145         (Action action, uint256 auctionId, address collateral, uint256 collateralAm, uint256
                maxPrice) = abi.decode(
146             data, (Action, uint256, address, uint256, uint256)
147         );
148         require(action == Action.NORMAL, "such action is not implemented");
149         uint256 tokenBalance = IERC20(token).balanceOf(address(this));
150         require(tokenBalance >= amount, "borrow amount not received");
151
152
153         auction.buyFromAuction(collateral, auctionId, collateralAm, maxPrice, address(this));
154         uint256 minOut = amount + fee - IERC20(token).balanceOf(address(this));
155
156
157         address[] memory path = new address[](2);
158         path[0] = collateral;
159         path[1] = token;
160         dex.swapExactTokensForTokens(IERC20(collateral).balanceOf(address(this)), minOut, path,
                address(this), block.timestamp + 300);
161         return keccak256("ERC3156FlashBorrower.onFlashLoan");
162     }
```

Listing 2.1: FlashBuy.sol

Impact The swapping process is vulnerable to sandwich attack, which leads to funds losses.

Suggestion When calculating `minOut`, the value to be repaid to the `flash loan` should be used directly without subtracting the existing token amount within the contract.

2.1.2 Potential DoS due to improper approve amount

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Function `flashBuyAuction()` first approves the quantity of collateral to be swapped from the `DEX` before initiating the `flash loan`. This approval is done in preparation for selling the collateral in the callback function `onFlashLoan()`. However, in the function `onFlashLoan()`, the actual quantity of collateral sold is the total amount of collateral present within the contract. This amount may be greater than the quantity of collateral bought during the flash loan, and the `DEX` will not have sufficient allowance for this large amount. This situation can result in a `DoS` vulnerability.

```
162     function flashBuyAuction(
163         address token,
164                 uint256 auctionId,
165                 uint256 borrowAm,
166                 address collateral,
167                 uint256 collateralAm,
168                 uint256 maxPrice
```

```
169         ) public {
170             require(borrowAm <= lender.maxFlashLoan(token));
171             bytes memory data = abi.encode(Action.NORMAL, auctionId, collateral, collateralAm,
                    maxPrice);
172             uint256 _fee = lender.flashFee(token, borrowAm);
173             uint256 _repayment = borrowAm + _fee;
174             uint256 _allowance = IERC20(token).allowance(address(this), address(lender));
175             IERC20(token).approve(address(lender), _allowance + _repayment);
176             IERC20(token).approve(address(auction), _allowance + _repayment);
177             IERC20(collateral).approve(address(dex), collateralAm);
178
179
180             lender.flashLoan(this, token, borrowAm, data);
181         }
```

Listing 2.2: FlashBuy.sol

```
128 function onFlashLoan(
129     address initiator,
130     address token,
131     uint256 amount,
132     uint256 fee,
133     bytes calldata data
134 ) external override returns (bytes32) {
135     require(
136         msg.sender == address(lender),
137         "FlashBorrower: Untrusted lender"
138     );
139     require(
140         initiator == address(this),
141         "FlashBorrower: Untrusted loan initiator"
142     );
143
144
145     (Action action, uint256 auctionId, address collateral, uint256 collateralAm, uint256
            maxPrice) = abi.decode(
146         data, (Action, uint256, address, uint256, uint256)
147     );
148     require(action == Action.NORMAL, "such action is not implemented");
149     uint256 tokenBalance = IERC20(token).balanceOf(address(this));
150     require(tokenBalance >= amount, "borrow amount not received");
151
152
153     auction.buyFromAuction(collateral, auctionId, collateralAm, maxPrice, address(this));
154     uint256 minOut = amount + fee - IERC20(token).balanceOf(address(this));
155
156
157     address[] memory path = new address[](2);
158     path[0] = collateral;
159     path[1] = token;
160     dex.swapExactTokensForTokens(IERC20(collateral).balanceOf(address(this)), minOut, path,
            address(this), block.timestamp + 300);
161     return keccak256("ERC3156FlashBorrower.onFlashLoan");
```

```
162     }
```

Listing 2.3: FlashBuy.sol

Impact Function `flashBuyAuction()` can never be executed successfully if the contract holds collateral.

Suggestion In `onFlashLoan()`, selling the exact amount of `collateralAm`.

2.1.3 Failed `transferFrom()` due to contract's lack of approval for owner

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `FlashBuy` contract, the `owner` of the contract can use the function `transferFrom()` to transfer any tokens held by the current contract address to any address. This function makes use of the standard `ERC20 transferFrom()` method for the transfer. However, there is no function implemented for approving the `owner` to transfer from the contract, thereby rendering the function `transferFrom()` ineffective for its intended use.

```
123     function transferFrom(address token) onlyOwner external {
124         IERC20(token).transferFrom(address(this), msg.sender, IERC20(token).balanceOf(address(this)
                ));
125     }
```

Listing 2.4: FlashBuy.sol

Impact The function `transferFrom()` in the `FlashBuy` contract cannot be used as intended.

Suggestion Replace function `transferFrom()` with function `transfer()`.

2.2 Note

2.2.1 Interaction with out-of-scope contracts

Introduced by `Version 1`

Description Within the current scope of the audit, there are certain functions that interact with contracts out of the audit scope, such as `AuctionProxy` as well as `DEX` in the `FlashBuy` contract. We assume that these contracts are secure and trusted.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS