Smart Contract Security Audit Report

Prepared for Lista DAO

Prepared by Supremacy

March 26, 2024

Contents

1 Introduction

Given the opportunity to review the design document and related codebase of the Lista DAO FlashBuy Contract, we outline in the report our systematic approach to evaluate potential security issues in the smart contract(s) implementation, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts can be further improved due to the presence of several issue related to either security or performance. This document outlines our audit results.

1.1 About Client

Lista DAO functions as the open-source decentralized stablecoin lending protocol powered by LSDfi. Users can undergo staking and liquid staking on Lista, as well as borrow lisUSD against a variety of decentralized collateral. Present on the BNB Chain, Lista aims to position lisUSD as the number one stablecoin in the crypto space, leveraging on innovative liquid staking solutions.

Item	Description
Client	Lista DAO
Website	https://lista.org
Type	Smart Contract
Languages	Solidity
Platform	EVM-compatible

1.2 Audit Scope

In the following, we show the Git repository of reviewed file and the commit hash used in this security audit:

- Repository: https://github.com/lista-dao/lista-dao-contracts/blob/develop/contracts
- Commit Hash: 3cb89728f9633d9d9392c4f822aba6beaf345b22

Below are the files in scope for this security audit and their corresponding MD5 hashes.

Filename	MD5
`./FlashBuy.sol`	`b41498f618f13d64a763db68a2008651`

1.3 Changelogs

Version	Date	Description
0.1	March 26, 2024	Initial Draft

1.4 About Us

Supremacy is a leading blockchain security firm, composed of industry hackers and academic researchers, provide top-notch security solutions through our technology precipitation and innovative research.

We are reachable at Twitter (https://twitter.com/SupremacyHQ), or Email (contact@supremacy.email).

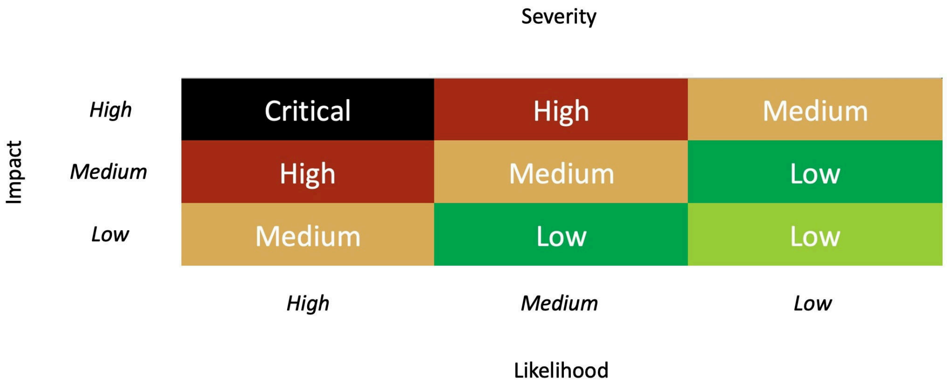
1.5 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- Likelihood represents the likelihood of a finding to be triggered or exploited in practice

- Impact specifies the technical and business-related consequences of a finding
- Severity is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.



Severity

Impact

	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

Likelihood

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding’s risk should always be closely checked, regardless of severity.

2 Findings

The table below summarizes the findings of the audit, including status and severity details.

ID	Severity	Description	Status
1	Medium	The potential risk of fund pool depletion	Undetermined
3	Low	Support of different ERC20 tokens	Undetermined
3	Informational	Lack of address validation	Undetermined
4	Informational	Redundant code removal	Undetermined

2.1 Medium

1. The potential risk of fund pool depletion [Medium]

```
Severity: Medium                    Likelihood: Medium                    Impact: Medium
Status: Undetermined
```

Description:

In the `FlashBuy::flashBuyAuction()` function, the contract call the lender contract's `flashLoan()` function. However, if the attempted repayment, the amount to be repaid is the debt (`borrowAm + _fee`). Also, if there is a certain amount of value in the contract, there may be a malicious consumption of the handling fee.

```
    /// @dev Initiate a flash loan
    function flashBuyAuction(
        address token,
        uint256 auctionId,
        uint256 borrowAm,
        address collateral,
        uint256 collateralAm,
        uint256 maxPrice
    ) public {
        require(borrowAm <= lender.maxFlashLoan(token));
        bytes memory data = abi.encode(Action.NORMAL, auctionId, collateral,
collateralAm, maxPrice);
        uint256 _fee = lender.flashFee(token, borrowAm);
        uint256 _repayment = borrowAm + _fee;
        uint256 _allowance = IERC20(token).allowance(address(this),
address(lender));
        IERC20(token).approve(address(lender), _allowance + _repayment);
        IERC20(token).approve(address(auction), _allowance + _repayment);
        IERC20(collateral).approve(address(dex), collateralAm);

        lender.flashLoan(this, token, borrowAm, data);
    }
```

FlashBuy.sol

Recommendation: Establish a access control mechanism.

2.2 Low

3. Support of different ERC20 tokens [Low]

`Severity: Low` `Likelihood: Low` `Impact: Low`

`Status: Undetermined`

Description:

The current version of the code does not handle special cases of tokens, such as return `true` / `false` tokens on success, nor does the current code check whether the transfer was successful.

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

FlashBuy.sol

Recommendation: Consider use the SafeERC20 library to handle token transfers.

2.3 Informational

4. Lack of address validation [Informational]

`Status: Undetermined`

Description:

The `FlashBuy::initialize()` function lack address validation. However, once set incorrectly, for example, set to zero address, it will result in denial of service.

```
    // --- Init ---
    function initialize(IERC3156FlashLender lender_, IAuctionProxy auction_, IDEX dex_) public initializer {
        __Ownable_init();

        lender = lender_;
        auction = auction_;
        dex = dex_;
    }
```

FlashBuy.sol

Recommendation: Add zero address validation.

4. Redundant code removal [Informational]

`Status: Undetermined`

Description:

In the `FlashBuy::flashBuyAuction()`, there is an process of initiating a flashloan, while the repayment process is initiated by the transfer of the `lender` contract. Therefore, the `lender` contract needs to be approved, but the approval amount doesn't apply here, `#175` is the only interface here that `approve` the lender, so if the `_allowance` is fully utilized each time, the `_allowance + _repayment` formula isn't needed.

```
161     /// @dev Initiate a flash loan
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         lender.flashLoan(this, token, borrowAm, data);
180     }
```

FlashBuy.sol

Recommendation:

Revise the `#L175` code logic as `IERC20(token).approve(address(lender), _repayment);`

3 Disclaimer

This security 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 security 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, also cannot make guarantees about any additional code added to the assessed project after the audit version. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contract(s). 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.