Security Audit Report for ListaToken

Date: April 3rd, 2024 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Lista
Target	ListaToken

Version History

Version	Date	Description
1.0	April 3rd, 2024	First Version

Signature

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

Email: audit@blocksec.com

Date: 2024-04-03 13:20:04+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 of ListaToken[1] of Lista. Note that, we did **NOT** audit all the modules in the repository. The modules covered by this audit report include `lista-token` folder contract only. Specifically, the files covered in this audit include:

```
ListaToken.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
ListaToken	Version 1	68e0ade64b1d401117428fd77b2b594006c6db15
	Version 2	381b736285db002bcb3396101fd74a7484f1a3fd

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] https://github.com/lista-dao/lista-token/blob/master/contracts/ListaToken.sol

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

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 **one** potential issue.

- High Risk: 0
- Medium Risk: 1
- Low Risk: 0
- Recommendation: 0
- Note: 0

ID	Severity	Description	Category	Status
1	Medium	Signature Replay Risk due to Hard Fork	DeFi Security	Fixed

The details are provided in the following sections.

2.1 DeFi Security

2.1.1 Signature Replay Risk due to Hard Fork

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Hard fork can occur in blockchain networks, as seen in the past with `Ethereum`. In the event of a hard fork, the `chainID` undergoes a change. If a contract continues to use the old `chainID`, the `DOMAIN_SEPARATOR` becomes invalid. Moreover, if the protocol is deployed across multiple chains, a valid signature can be replayed on a different chain at a later time.

```
 constructor(address owner) ERC20(_NAME, _SYMBOL) {
   bytes32 hashedName = keccak256(bytes(_NAME));
   bytes32 hashedVersion = keccak256(bytes(EIP712_VERSION));
   DOMAIN_SEPARATOR = keccak256(
       abi.encode(
           EIP712_DOMAIN,
           hashedName,
           hashedVersion,
           block.chainid,
           address(this)
       )
   );


   // mint 1B tokens to the lista treasury account
   _mint(owner, 1_000_000_000 * 10 ** decimals());
}
```

Listing 2.1: ListaToken.sol

Impact The `DOMAIN_SEPARATOR` will become invalid due to the hard fork of the chain. What's worse, the valid signature can be replayed on different chains.

Suggestion I Implement relevant logic to enable the contract to update the `DOMAIN_SEPARATOR` based on the latest `chainID`.

BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS