HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: LTO Network
Date: September 16th, 2021

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for LTO Network.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	ERC20 token; Token Sale
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/ltonetwork/lto-erc20-token
Commit	0f439d8b1f14f4a81a89bc0b2ac210e7340e848e
Technical Documentation	YES
JS tests	YES
Timeline	14 SEPTEMBER 2021 - 16 SEPTEMBER 2021
Changelog	16 SEPTEMBER 2021 - Initial Audit

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Introduction 4

Scope 4

Executive Summary 5

Severity Definitions 7

AS-IS overview **Ошибка! Закладка не определена.**

Audit overview 8

Conclusion 11

Disclaimers 13

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by LTO Network (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contract and its code review conducted between September 14th, 2021 - September 16th, 2021.

Scope

The scope of the project is smart contracts in the repository:

Repository:
https://github.com/ltonetwork/lto-erc20-token
Commit:
0f439d8b1f14f4a81a89bc0b2ac210e7340e848e
Technical Documentation: Yes
JS tests: Yes
Contracts:
BalanceCopier.sol
ERC20PreMint.sol
FakeWallet.sol
LTOToken.sol
LTOTokenSale.sol
Migrations.sol

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ DoS with (Unexpected) Throw ▪ DoS with Block Gas Limit ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ Costly Loop ▪ ERC20 API violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency ▪ Data Consistency

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

According to the assessment, the Customer's smart contracts are secured.

Insecure	Poor secured	Secured	Well-secured

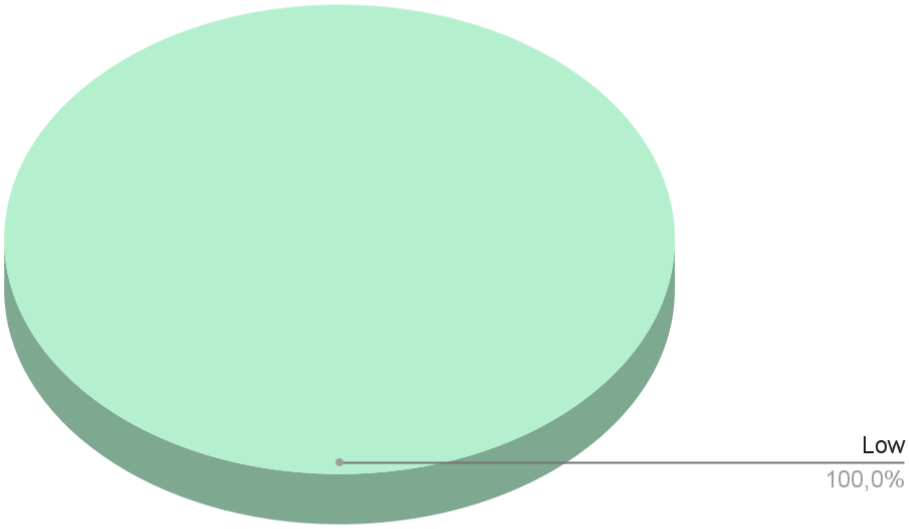
You are here 

Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. All found issues can be found in the Audit overview section.

As a result of the audit, security engineers found **5** low severity issues.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Graph 1. The distribution of vulnerabilities after the audit.



Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that can't have a significant impact on execution

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Audit overview

Critical

No critical issues were found.

High

No high severity issues were found.

Medium

No medium severity issues were found.

Low

1. Two of eighteen tests are failing

   There are 18 tests for the project repository and 2 of them are failed, both with the same error: “Cannot read property 'status' of undefined”.

   Recommendation: Please fix failed tests and also ensure that your tests are cover at least 95% of branches.

```
16 passing (1s)
2 failing

1) Contract: LTOToken
     created token
       pre-minting tokens
         should not be possible to transfer while pre-minting:
   TypeError: Cannot read property 'status' of undefined
    at Context.it (test/LTOToken.test.js:75:34)
    at process._tickCallback (internal/process/next_tick.js:68:7)

2) Contract: LTOToken
     created token
       bridge
         when adding an intermediate addresses from a non bridge address
           should throw an error:
   TypeError: Cannot read property 'status' of undefined
    at Context.it (test/LTOToken.test.js:116:36)
    at process._tickCallback (internal/process/next_tick.js:68:7)



npm ERR! Test failed.  See above for more details.
```

2. Conformance to Solidity naming conventions

   Solidity defines a naming convention that should be followed. e.g. constants should be declared UPPER_CASE_WITH_UNDERSCORES.

   Recommendation: Follow the Solidity naming convention.

Lines: LTOTokenSale.sol#17-21

```
uint256 constant minimumAmount = 0.1 ether;
uint256 constant maximumCapAmount = 40 ether;
uint256 constant ethDecimals = 1 ether;
uint256 constant ltoEthDiffDecimals = 10**10;
uint256 constant bonusRateDivision = 10000;
```

3. Excess require check

   While using "Safemath.sub" function which is already checking that subtrahend is less than minuend, so putting additional require statement for the same comparison is excess.

   Recommendation: Please remove excess require statement.

Lines: LTOToken.sol#74-76

```
require(value <= bridgeBalance);

bridgeBalance = bridgeBalance.sub(value);
```

Lines: SafeMath.sol#40-45

```
function sub(uint256 a, uint256 b) internal pure returns (uint256) {
 require(b <= a);
 uint256 c = a - b;

 return c;
}
```

4. Implicit visibility declaration

   State variables that don't have explicitly declared visibility is defaulted as the internal, which is not always understandable for reviewers and could cause misunderstanding.

   Recommendation: Please try always to declare visibility explicitly.

Lines: LTOTokenSale.sol#17-21

```
uint256 constant minimumAmount = 0.1 ether;
uint256 constant maximumCapAmount = 40 ether;
uint256 constant ethDecimals = 1 ether;
uint256 constant ltoEthDiffDecimals = 10**10;
uint256 constant bonusRateDivision = 10000;
```

Lines: LTOToken.sol#11-12

```
uint8 constant PENDING_BRIDGE = 1;
uint8 constant PENDING_CONFIRM = 2;
```

Lines: BalanceCopier.sol#15

```
mapping (address => bool) copied;
```

5. A public function that could be declared external

public functions that are never called by the contract should be declared **external** to save gas.

Recommendation: Use the external attribute for functions never called from the contract.

Lines: BalanceCopier.sol#28

```
function copy(address _holder) public whenBothPaused {
```

Lines: BalanceCopier.sol#37

```
function copyAll(address[] _holders) public whenBothPaused {
```

Lines: BalanceCopier.sol#50

```
function done() public onlyOwner {
```

Lines: LTOTokenSale.sol#122

```
function startSale(uint256 _startTime, uint256 _rate, uint256 duration,
```

LTOTokenSale.sol#141

```
function getPurchaserCount() public view returns(uint256) {
```

LTOTokenSale.sol#164

```
function getPublicSaleInfo(address purchaser) public view returns
(uint256, uint256, uint256) {
```

LTOTokenSale.sol#169

```
function () payable public {
```

LTOTokenSale.sol#231

```
function withdrawal() public onlyUserWithdrawalTime {
```

LTOTokenSale.sol#241

```
function clear(uint256 tokenAmount, uint256 etherAmount) public
purchasersAllWithdrawn onlyClearTime onlyOwner {
```

LTOTokenSale.sol#250

```
function withdrawFailed(address alternativeAddress) public
onlyUserWithdrawalTime nonReentrant {
```

LTOTokenSale.sol#261

```
function addCapFreeAddress(address capFreeAddress) public
onlyCapListAddress {
```

LTOTokenSale.sol#267

```
function removeCapFreeAddress(address capFreeAddress) public
onlyCapListAddress {
```

LTOTokenSale.sol#277

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

```
function currentBonus() public view returns(uint256) {
```

LTOToken.sol#32

```
function addIntermediateAddress(address _intermediate) public onlyBridge
{
```

LTOToken.sol#42

```
function confirmIntermediateAddress() public {
```

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

As a result of the audit, security engineers found **5** low severity issue.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only – we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.