verichains

SECURITY AUDIT OF

LAUNCHZONE TOKEN SMART CONTRACT

LAUNCH ZONE

PUBLIC REPORT

SEP 20, 2021

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ACRONYMS AND ABBREVIATIONS

NAME	DESCRIPTION
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network
Binance Chain	Binance Chain is a blockchain software system developed by Binance and its community.
Binance Smart Chain (BSC)	Binance Smart Chain (BSC) is a blockchain network built for running smart contract-based applications. BSC runs in parallel with Binance's native Binance Chain (BC), which allows users to get the best of both worlds: the high transaction capacity of BC and the smart contract functionality of BSC.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Sep 20, 2021. We would like to thank the LaunchZone team for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the LaunchZone Token Contract. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified several vulnerable issues in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About LaunchZone and LaunchZone Token

LaunchZone is one of the most integrated Defi ecosystems on Binance Smart Chain where you can farming, staking, swapping, etc. They are working hard to bring utilities to other blockchains and create more economic opportunities for users.

LaunchZone Token (LZ) is the BEP-20 token of LaunchZone, with a maximum total supply of 50 million.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the LaunchZone Token Contract.

The audited contract is the LaunchZone Token Contract that was deployed on Binance Smart Chain Mainnet at address *0x3b78458981eb7260d1f781cb8be2caac7027dbe2*. The details of the deployed smart contract are listed in Table 1.

FIELD	VALUE
Contract Name	LZToken
Contract Address	0xfec70a44c683cc86397abed3700e115f0e956505
Compiler Version	v0.6.12+commit.27d51765
Optimization Enabled	Yes with 200 runs
Explorer	https://bscscan.com/address/0x3b78458981eb7260d1f781cb8be2caac7027dbe2

Table 1: The deployed smart contract details

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Integer Overflow and Underflow

- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in Table 2, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 2: Vulnerability severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The LaunchZone Token Contract is a BEP-20 Token Contract[1], which implements a standard interface for token as defined in IBEP-20[2]. This standard provides basic functionality to transfer tokens, as well as allow tokens to be approved so they can be spent by another on-chain third party.

Table 3 lists some properties of the audited LaunchZone Token Contract (as of the report writing time).

PROPERTY	VALUE
Name	LaunchZone
Symbol	LZ
Decimals	18
Owner	0xdad254728a37d1e80c21afae688c64d0383cc307
Maximum Total Supply	50,000,000 ($\times 10^{18}$) Note: the number of decimals is 18, so the maximum total representation tokens will be 50,000,000, or 50 million.

Table 3: The LaunchZone Token properties

Besides the standard interface of an BEP-20 token, the LaunchZone Token Contract also implements some additional functional logics by extending the following contracts:

- Context: provides information about the current execution context, including the sender of the transaction and its data. While these are generally available via msg.sender and msg.data, they should not be accessed in such a direct manner, since when dealing with GSN meta-transactions the account sending and paying for execution may not be the actual sender (as far as an application is concerned).
- Ownable: this contract has a basic access control mechanism, where there is an account (an owner) that can be granted exclusive access to specific functions. By default, the owner account will be the one that deploys the contract. The current owner can renounce or transfer ownership to a new owner.
- Pausable: this contract allows children to implement an emergency stop mechanism that can be triggered by an authorized account.

[1] https://docs.binance.org/smart-chain/developer/BEP20.html
[2] https://docs.binance.org/smart-chain/developer/IBEP20.sol

- ReentrancyGuard: this abstract contract helps prevent reentrant calls to a function.

2.2. Contract codes

The LaunchZone Token Contract was written in Solidity language[3], with the required version to be `0.6.12`.

The source codes consist of six interfaces, three libraries, one abstract contract and five contracts. Source codes in the LaunchZone Token Contract are extended from Binance's implementation template of BEP20-related contracts[4] and OpenZeppelin Contracts[5] with some additional modifications.

2.2.1. IBEP20 interface

This is the interface of the BEP-20 token standard. The source code is referenced from the official Binance's documentation.

2.2.2. ILZ interface

This is the interface of the LZ token which be extended from the IBEP20 interface to provides some additional functions.

2.2.3. IUniswapV2Factory interface

This is the interface to interact with Uniswap V2 Factory contract which is referenced from Uniswap's repository.

2.2.4. IUniswapV2Pair interface

This is the interface to interact with Uniswap V2 Pair contract which is referenced from Uniswap's repository.

2.2.5. IUniswapV2Router01 interface

This is the interface to interact with Uniswap V2 Router01 contract which is referenced from Uniswap's repository.

2.2.6. IUniswapV2Router02 interface

This is the interface to interact with Uniswap V2 Router02 contract which is referenced from Uniswap's repository.

[3] https://docs.soliditylang.org/
[4] https://docs.binance.org/smart-chain/developer/BEP20Token.template
[5] https://openzeppelin.com/contracts/

2.2.7. SafeERC20 library

This library provides wrappers around ERC20 operations that throw on failure (when the token contract returns false). Tokens that return no value (and instead revert or throw on failure) are also supported, non-reverting calls are assumed to be successful. The source code is referenced from OpenZeppelin's implementation.

2.2.8. SafeMath library

This library provides wrappers over Solidity's arithmetic operations with added overflow checks. The source code is referenced from OpenZeppelin's implementation.

2.2.9. Address library

This library provides a collection of functions related to the address type. The source code is referenced from OpenZeppelin's implementation.

2.2.10. ReentrancyGuard abstract contract

This abstract contract helps prevent reentrant calls to a function. Inheriting from `ReentrancyGuard` will make the ***`nonReentrant`*** modifier available, which can be applied to functions to make sure there are no nested (reentrant) calls to them. The source code is referenced from OpenZeppelin's implementation.

2.2.11. Ownable contract

This contract makes the inherited token contract ownable. The source code is referenced from OpenZeppelin's implementation.

2.2.12. Context contract

This contract provides information about the current execution context, including the sender of the transaction and its data. The source code is referenced from OpenZeppelin's implementation.

2.2.13. Pausable contract

This contract allows children to implement an emergency stop mechanism that can be triggered by an authorized account. The source code is referenced from OpenZeppelin's implementation.

2.2.14. BEP20 contract

This contract is a standard BEP20 token which is referenced from the official Binance's documentation.

2.2.15. LZToken contract

This is the main contract of LaunchZone Token Contract, which extends the `BEP20`, `Context` and `Ownable` contracts. Below is a summary of some important functions in this contract:

- ***`constructor()`***: specifies the name, symbol and setup the initial value for the contract variables.
- ***`stake(address token, uint256 amount)`***: stake an ***`amount`*** of the input ***`token`***.
- ***`unstake(address token, uint256 amount)`***: unstake an ***`amount`*** of the input ***`token`***.
- ***`claimUnlocked(address token)`***: claim unlocked reward tokens based on the staked amount of the input ***`token`***.
- ***`mintUnlockedToken(address to, uint256 amount)`***: allows authorized users to mint ***`amount`*** of unlocked reward tokens to address ***`to`***.
- ***`mintLockedToken(address to, uint256 amount)`***: allows authorized users to mint ***`amount`*** of locked reward tokens to address ***`to`***.
- ***`_transfer(address sender, address recipient, uint256 amount)`***: override the standard _transfer function with fee, locked amount and delegated votes calculation.
- ***`swapAndLiquify(uint256 contractTokenBalance)`***: swaps half of the current balance of the LaunchZone token contract and adds liquidity to the LZToken – WETH pool.
- ***`delegate(address delegatee)`***: delegates the token balance to the ***`delegatee`***.
- ***`delegates(address delegator)`***: get the current ***`delegate`*** votes of the delegatee at ***`delegator`*** address.

2.3. Findings

This section contains a detailed analysis of all the vulnerabilities that were discovered by the audit team during the audit process.

2.3.1. LZToken – Fee is not updated for `_delegates` in `_transfer` function

~~[MEDIUM]~~ INFORMATIVE

When transferring tokens with fee, the ***`delegates`*** mapping is not updated accordingly. Specifically, the ***`delegates`*** of the sender is not decreased by fee.

```
function _transfer(address sender, address recipient, uint256 amount) internal virt
ual whenNotPaused override{
    if (_isExcludedFromFee[sender] != true && _isExcludedFromFee[recipient] != true
) {
        uint256 _liquidityAmount = 0;
        uint256 _stakingAndBurnAndTeamAmount = 0;
        if (_liquidityFee > 0) {
            _liquidityAmount = amount.mul(_liquidityFee).div(100);
            // Take Liquidity
            super._transfer(sender, address(this), _liquidityAmount);
            _unlockTransfer(sender, address(this), _liquidityAmount);
            // Check if _liquidityReservoir is reached then swap
            uint256 contractTokenBalance = balanceOf(address(this));
            if (
                contractTokenBalance >= _liquidityReservoir &&
                !inSwapAndLiquify
            ) {
                swapAndLiquify(contractTokenBalance);
            }
        }
        if (_stakingAndBurnAndTeamAlloc > 0) {
            _stakingAndBurnAndTeamAmount = amount.mul(_stakingAndBurnAndTeamAlloc).
div(100);
            super._transfer(sender, _foundation, _stakingAndBurnAndTeamAmount);
            _unlockTransfer(sender, _foundation, _stakingAndBurnAndTeamAmount);
        }
        amount = amount.sub(_liquidityAmount).sub(_stakingAndBurnAndTeamAmount);
    }

    super._transfer(sender, recipient, amount);
    _unlockTransfer(sender, recipient, amount);
    _moveDelegates(_delegates[sender], _delegates[recipient], amount);
}
```

RECOMMENDATION

To fix this issue, we must call ***_moveDelegates*** function to update the ***delegates*** mapping when transferring fee.

```
function _transfer(address sender, address recipient, uint256 amount) internal virt
ual whenNotPaused override{
    if (_isExcludedFromFee[sender] != true && _isExcludedFromFee[recipient] != true
) {
        uint256 _liquidityAmount = 0;
        uint256 _stakingAndBurnAndTeamAmount = 0;
        if (_liquidityFee > 0) {
            _liquidityAmount = amount.mul(_liquidityFee).div(100);
            // Take Liquidity
            super._transfer(sender, address(this), _liquidityAmount);
            _unlockTransfer(sender, address(this), _liquidityAmount);

            _moveDelegates(_delegates[sender], _delegates[address(this)], _liquidit
yAmount);

            // Check if _liquidityReservoir is reached then swap
            uint256 contractTokenBalance = balanceOf(address(this));
            if (
                contractTokenBalance >= _liquidityReservoir &&
                !inSwapAndLiquify
            ) {
                swapAndLiquify(contractTokenBalance);
            }
        }
        if (_stakingAndBurnAndTeamAlloc > 0) {
            _stakingAndBurnAndTeamAmount = amount.mul(_stakingAndBurnAndTeamAlloc).
div(100);
            super._transfer(sender, _foundation, _stakingAndBurnAndTeamAmount);
            _unlockTransfer(sender, _foundation, _stakingAndBurnAndTeamAmount);

            _moveDelegates(_delegates[sender], _delegates[_foundation], _stakingAnd
BurnAndTeamAmount);
        }
        amount = amount.sub(_liquidityAmount).sub(_stakingAndBurnAndTeamAmount);
    }

    super._transfer(sender, recipient, amount);
    _unlockTransfer(sender, recipient, amount);
    _moveDelegates(_delegates[sender], _delegates[recipient], amount);
}
```

UPDATES

This finding has been acknowledged by LaunchZone team. However, the delegate function will not be used, so we've changed the severity of this issue to INFORMATIVE.

2.4. Additional notes and recommendations

2.4.1. LZToken – Improve lockTheSwap modifier

The ***lockTheSwap*** modifier is being used in ***swapAndLiquify*** function. But when calling the ***swapAndLiquify*** function, we must check the ***inSwapAndLiquify*** variable.

```
modifier lockTheSwap {
    inSwapAndLiquify = true;
    _;
    inSwapAndLiquify = false;
}

function _transfer(address sender, address recipient, uint256 amount) internal virt
ual whenNotPaused override{
    // ...

    if (
        contractTokenBalance >= _liquidityReservoir &&
        !inSwapAndLiquify
    ) {
        swapAndLiquify(contractTokenBalance);
    }

    // ...
}
```

RECOMMENDATION

We can improve the ***lockTheSwap*** modifier as below to skip checking the ***inSwapAndLiquify*** variable as below.

```
modifier lockTheSwap {
    if (!inSwapAndLiquify) {
        inSwapAndLiquify = true;
        _;
        inSwapAndLiquify = false;
    }
}

function _transfer(address sender, address recipient, uint256 amount) internal virt
ual whenNotPaused override{
    // ...

    if (
        contractTokenBalance >= _liquidityReservoir
    ) {
        swapAndLiquify(contractTokenBalance);
    }

    // ...
}
```

UPDATES

This finding has been acknowledged by LaunchZone team.

2.4.2. LZToken – Check zero amount before stake

In ***stake*** function, when the staking amount is zero, we can skip all the remainder logic for gas saving.

```
function stake(address token, uint256 amount) external override nonReentrant return
s (bool) {
    require(_unlockFactor[token] > 0, "LZ: FACTOR_NOT_SET");
    require(_unlockBlockGap[token] > 0, "LZ: BLOCK_GAP_NOT_SET");
    _pullToken(token, _msgSender(), amount);
    LpStakeInfo storage info = _stakingRecords[_msgSender()][token];
    uint256 unlockedAmount = _settleUnlockAmount(_msgSender(), token, info.amountSt
aked, info.blockNumber);
    _updateStakeRecord(_msgSender(), token, info.amountStaked.add(amount));
    _mintUnlocked(_msgSender(), unlockedAmount);
    emit LOG_STAKE(_msgSender(), token, amount);
    return true;
}
```

RECOMMENDATION

Update the ***stake*** function as below.

```
function stake(address token, uint256 amount) external override nonReentrant return
s (bool) {
    require(_unlockFactor[token] > 0, "LZ: FACTOR_NOT_SET");
    require(_unlockBlockGap[token] > 0, "LZ: BLOCK_GAP_NOT_SET");

    // FIX HERE
    require(amount > 0, "LZ: ZERO_STAKE_AMOUNT");

    _pullToken(token, _msgSender(), amount);
    LpStakeInfo storage info = _stakingRecords[_msgSender()][token];
    uint256 unlockedAmount = _settleUnlockAmount(_msgSender(), token, info.amountSt
aked, info.blockNumber);
    _updateStakeRecord(_msgSender(), token, info.amountStaked.add(amount));
    _mintUnlocked(_msgSender(), unlockedAmount);
    emit LOG_STAKE(_msgSender(), token, amount);
    return true;
}
```

UPDATES

This finding has been acknowledged by LaunchZone team.

3. AUDIT RESULT

Version	Date	Status/Changes	Created by
1.0	Sep 20, 2021	Initial private report	Verichains Lab
1.1	Sep 20, 2021	Public report	Verichains Lab

APPENDIX A: FUNCTION CALL GRAPH

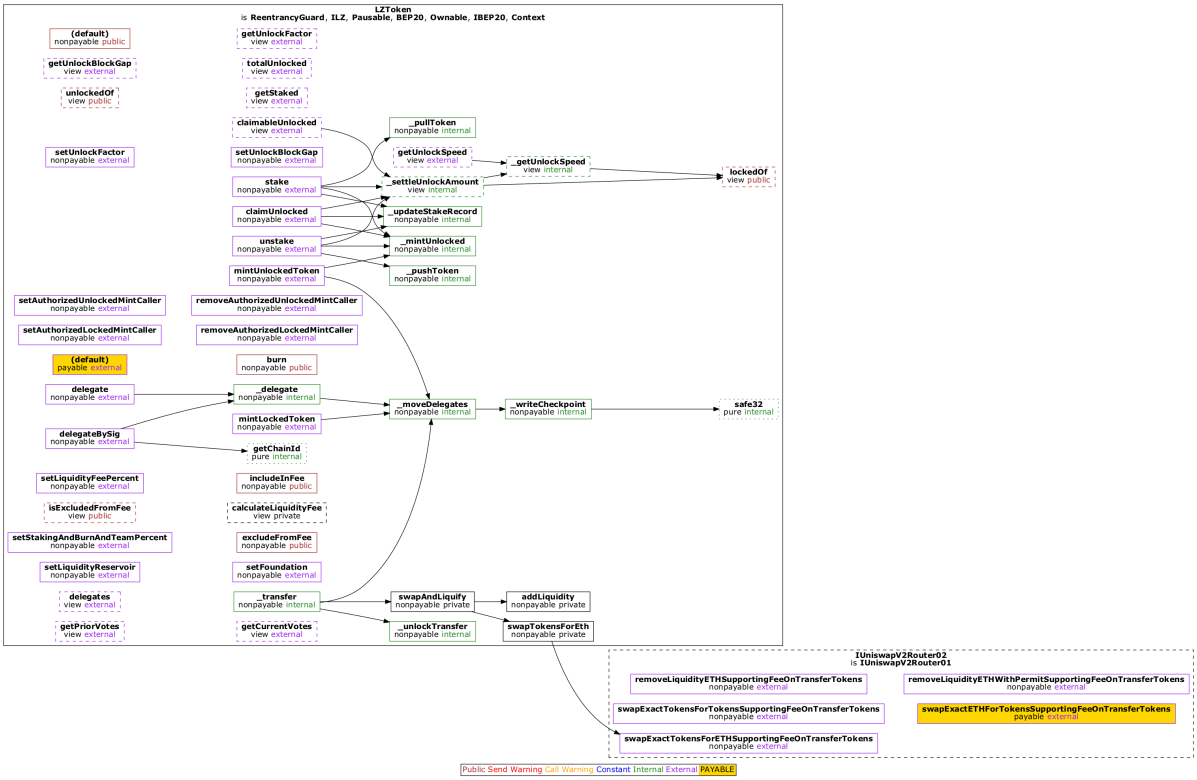

Figure 1: The function call graph of LZToken smart contract