SECURITY AUDIT OF

LOTTY STAKING CONTRACT

Public Report

Aug 10, 2023

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

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.
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.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Aug 10, 2023. We would like to thank the LOTTY 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 LOTTY Staking 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 some vulnerable issues in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About LOTTY Staking Contract

LOTTY is the ultimate token with massive utility for everyone. All tax proceeds will be used to fund LOTTY operations and purchase real world lottery tickets from 18 countries around the world. All winnings will be distributed back to token holders (per snapshot).

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of LOTTY Staking Contract. It was conducted on commit `d97ced03548a5f47ec4bb4fc2cdb390fef4d3ee9` from git repository link: *https://github.com/Lotty-Lotto/LOTTY-ERC20-Smart-Contract*.

There is a file in our audit scope. The latest version of following files were made available in the course of the review:

SHA256 Sum	File
fa083f80cb663f49ae93eac30945daa98b43b7d8bf09b775bf80bfca88eb3265	LottyStaking.sol

Table 1. The audit files

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 were 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 below, 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. 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 LOTTY Staking Contract was written in `Solidity` language, with the required version to be `^0.8.21;`.

2.1.1. Threat Model

Owner of the contract

The `LottyStaking` inherits `Ownable` that assigns deployer is owner of the contract. However, the owner is not granted permission to engage in specialized interactions with the contract.

Users

The contract allows users to stake ETH and LOTTY tokens and users can unstake them.

- `stake()`: Users must send ETH and amount of LOTTY desired, and then the contract adds liquidity to the ETH/LOTTY pair via the UniswapV2 Router. The staking (`StakePosition`) is stored in a struct including sender (owner of staking), amount of liquidity, staking time, and unique `nonce`. ***Note: Only EOA accounts can call the function. User can decide `timeLock`.***
- `unstake()`: Users unstake a `StakePosition` with a corresponding `nonce` and require the sender to be the owner of the stake. The function removes the liquidity equivalent amount in `StakePosition` via the UniswapV2 Router.

2.2. Findings

During the audit process, the audit team found some additional recommendations in the given version of the LOTTY Staking Contract.

2.2.1. [INFORMATIVE] Saving gas by directly remove liquidity to `msg.sender`

Position:

- `LottyStaking.sol`#L181

2.2.1.1.Description

The `unstake()` function removes liquidity and sends an amount of tokens/ETHs to the contract address. After that, the contract sends the sender the whole them. We suggest using the 'to' argument with the value `msg.sender` in `removeLiquidityETH`.

```
function unstake(
    //...
) public {
    //...
    uniswapV2Router.removeLiquidityETH(
        address(lotty),
        position.liquidity,
        _amountLottyMin,
        _amountETHMin,
-        address(this),
+       msg.sender
        _deadline
    );

-    uint256 balance = address(this).balance;
-    if (balance > 0) {
-        (bool callSuccess, ) = payable(msg.sender).call{value: balance}("");
-        if (!callSuccess) revert BalanceTransferFailed();
-    }

-    uint256 lottyBalance = lotty.balanceOf(address(this));
-    if (lottyBalance > 0) {
-        lotty.transfer(msg.sender, lottyBalance);
-    }

    emit Unstake(
        msg.sender,
        _positionNonce,
        block.timestamp,
        position.liquidity
    );
}
```

UPDATES

- ***Aug 10, 2023***: This issue has been acknowledged and fixed by the LOTTY team in commit `1e6e6c16e0fd97fb04c471a2053660f344a93a35`.

2.2.2. [INFORMATIVE] Unnecessary usage of `Ownable` contract

2.2.2.1. Description

The ownership concept is not utilized within the contract's logic, so the inclusion of the `Ownable` contract is unnecessary. We recommend removing the `Ownable` import and the inheritance from `LottyStaking` contract.

UPDATES

- ***Aug 10, 2023***: This issue has been acknowledged. `Ownable` will be applied to the upcoming features.

APPENDIX

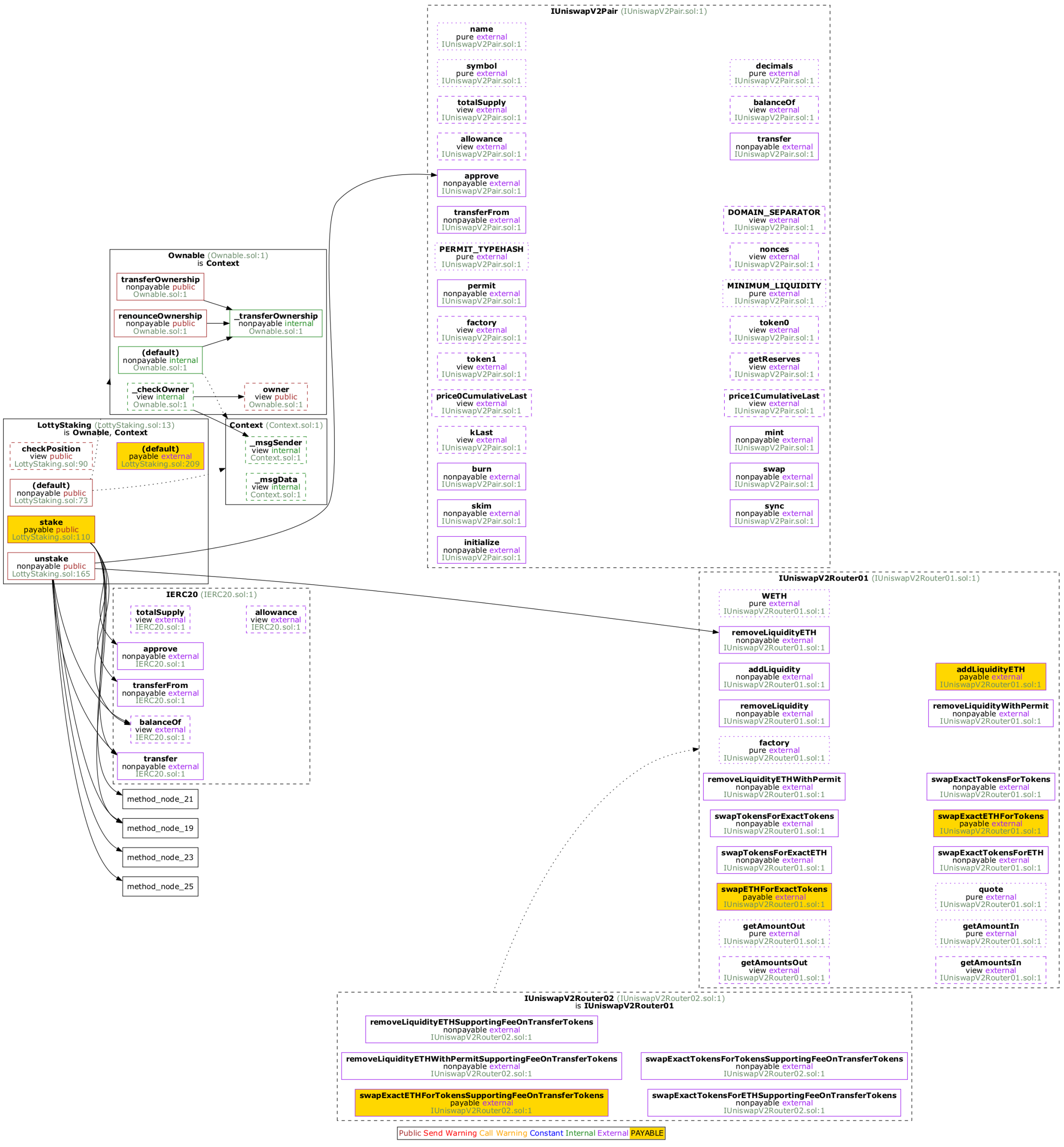

Image 1. LottyStaking contract call graph

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Aug 08, 2023*	Public Report	Verichains Lab
1.1	*Aug 10, 2023*	Public Report	Verichains Lab

Table 3. Report versions history