QuillAudits

Audit Report
November, 2022

For

LAST MAN STANDING

Table of Content

Executive Summary

Project Name	Last Man Standing
Overview	Last Man Standing is a gamified way of staking a contract which incentivizes the player of the game who stakes at last. If nobody has staked in the last 24 hours then rewards are transferred to the last staker of the contract. Also staking has its benefits because stakers also get staking rewards based on the staked amount.
Timeline	15 November, 2022 - 22 November, 2022.
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyze Last Man Standing Contract codebase for quality, security, and correctness. *https://github.com/part-time-small-brain/Last-Man-Standing* Commit hash: 923f7c160f428b5e60f0b4128edef4487286364c

High
Medium
Low
Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	1	1

Types of Severities

High

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

A. Contract - Last Man Standing

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

A1. Hardcoded address in constructor

Description

In token contract address devWallet is hardcoded in constructor which can cause issues in staking contract because it is used to supply/transfer tokens for rewards to the staking contract. But the function which is calling it is using onlyOwner modifier which suggests that it can only be called by owner If deployer's address is not passed as devWallet to token address then it can cause issues.

Remediation

To remediate the issue please either remove the address from the constructor and add another function to set wallet or just make sure you're providing the correct address as devWallet.

Status

Resolved

Informational Issues

A2. Incomplete TODO’s in contract

Description

There are incomplete TODO’s in code mentioning remaining functionality. One TODO above calculateReward() function suggests that the function should be called twice a day using a CRON job but because of the condition in calculateReward() function we can only call it once.

Remediation

Before shipping the final code please make sure to complete all the TODO’s in the contract

Status

Acknowledged

A.3 General Recommendation

Description

Variables owner and erc20Contract can be immutable as they are set in constructor and won’t be changing
Most of the functions such as transferAccidentallyLockedTokens(), addRewards(), setAPR(), setRewardTimeStamp(), enableStaking() can be set as external to save gas.
checkIfStakerExist when declared can be kept without assigning the value on line 95 can save gas as default value is always false.
Require statement comments can be shortened to save gas.
Function name is wrong: caclulateRewards() should be calculateRewards()

Status

Resolved

Functional Testing

Last Man Standing

- [PASS] testBurn() (gas: 84630)
- [PASS] testCompoundRewards() (gas: 8624945)
- [PASS] testEnableStaking() (gas: 28859)
- [PASS] testFailRewardFunctionalityCheckTwiceIn24Hrs() (gas: 388857)
- [PASS] testFailUnstake() (gas: 51110)
- [PASS] testGetter() (gas: 21195)
- [PASS] testMinStakingAmount() (gas: 35603)
- [PASS] testRewardFunctionality() (gas: 387182)
- [FAIL. Reason: Arithmetic over/underflow]
- testFailRewardFunctionalityWhenUserHasMoreTokens() (gas: 733596)
- [PASS] testStakeToken() (gas: 217293)
- [PASS] testStakeTokensWithUsers() (gas: 542942)
- [PASS] testTransfer() (gas: 70632)
- [PASS] testUnStakeTokensWithUsers() (gas: 533741)
- [PASS] testUnstakeTokens() (gas: 207625)

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the Last Man Standing Contract. We performed our audit according to the procedure described above.

Some issues of Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Last Man Standing Platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Last Man Standing Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

700+
Audits Completed

$15B
Secured

700K
Lines of Code Audited

Follow Our Journey

Audit Report
November, 2022

For

LAST MAN STANDING

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com