QuillAudits

Audit Report
December, 2024

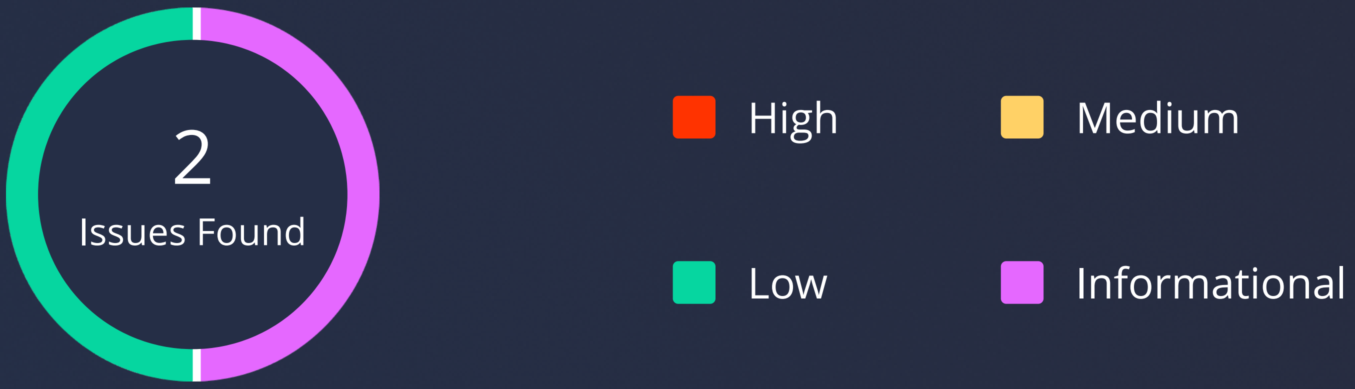
For

Table of Content

Executive Summary

Project Name	W Chain
Project URL	*https://w-chain.com/*
Overview	This is a staking contract where users can stake their eth above threshold value to become validators and choose to support the chain.
Audit Scope	The Scope of the Audit is to Analyse the Security,Code Quality and Correctness of W Chain Staking Contract.
Contracts In Scope	Staking.sol
Commit Hash	868d69ce6210cfbd466e0037a87d19c9b177a350
Language	Solidity
Blockchain	EVM
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	18th November 2024 - 27th November 2024
Updated Code Received	3rd December 2024
Review 2	3rd December 2024
Fixed In	*https://github.com/wadzchain/staking-contracts/commit/eb9679cb8baebb78779c203b988a3a634428d39a*

Number of Security Issues per Severity



	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

Checked Vulnerabilities

- ✓ Access Management
- ✓ Arbitrary write to storage
- ✓ Centralization of control
- ✓ Ether theft
- ✓ Improper or missing events
- ✓ Logical issues and flaws
- ✓ Arithmetic Correctness
- ✓ Race conditions/front running
- ✓ SWC Registry
- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Use of tx.origin
- ✓ Malicious libraries
- ✓ Compiler version not fixed
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ ERC's conformance
- ✓ Dangerous strict equalities
- ✓ Tautology or contradiction
- ✓ Return values of low-level calls
- ✓ Missing Zero Address Validation
- ✓ Private modifier
- ✓ Revert/require functions
- ✓ Multiple Sends
- ✓ Using suicide
- ✓ Using delegatecall
- ✓ Upgradeable safety
- ✓ Using throw

Checked Vulnerabilities

- ✓ Using inline assembly
- ✓ Style guide violation
- ✓ Unsafe type inference
- ✓ Implicit visibility level

Techniques and Methods

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

- The overall quality of code.
- Use of best practices.
- Code documentation and comments, match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC 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 analyzed 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

A 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 analyzed, 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 behavior 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, Foundry, Solhint, Mythril, Slither, Solidity statistic analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

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 Severity Issues

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 Severity Issues

Low-level severity issues can cause minor impact and 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 four 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.

Low Severity Issues

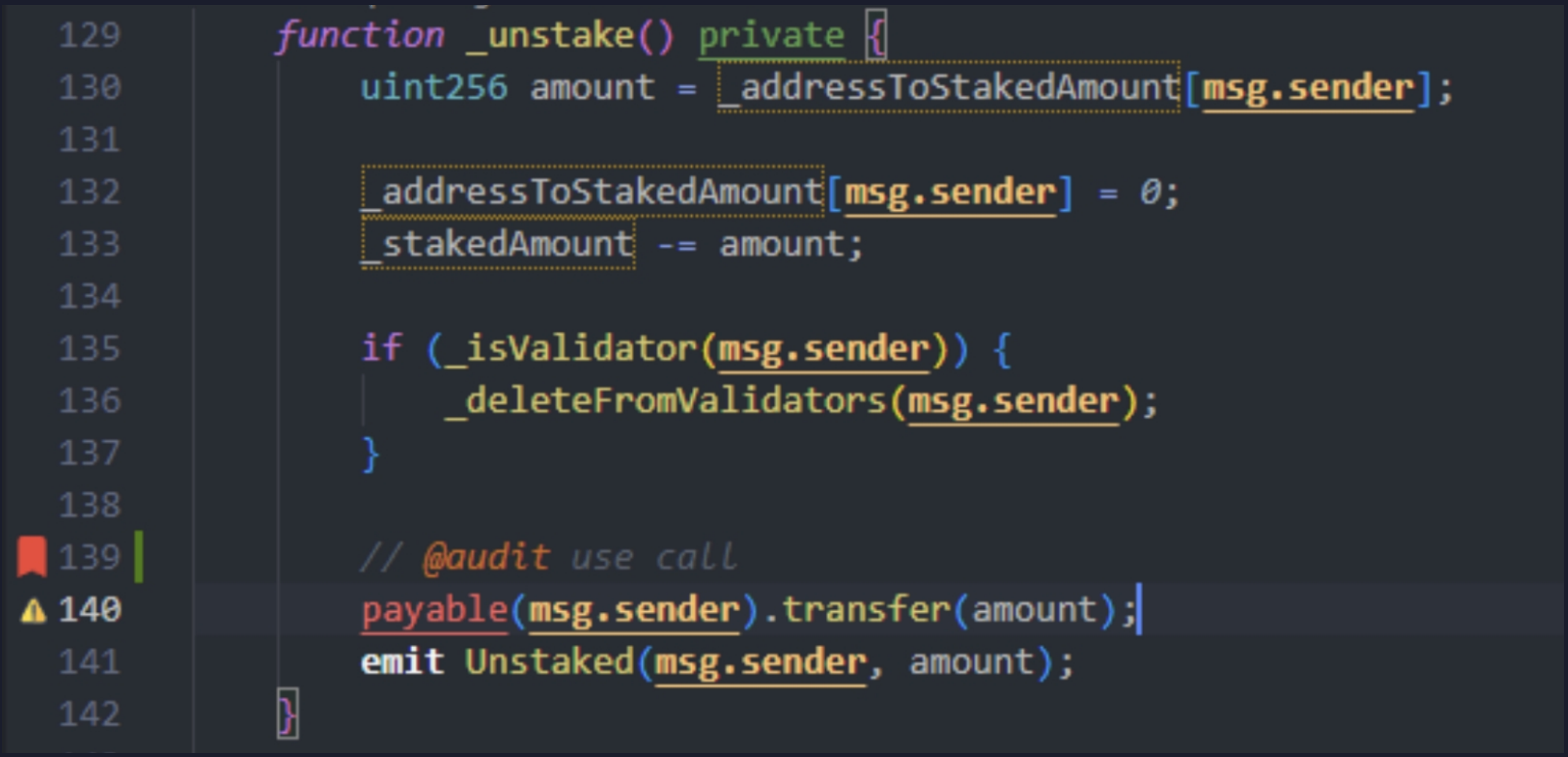
1. Use call instead of transfer

Path

staking.sol

Function name

stake()



```
    function _unstake() private {
        uint256 amount = _addressToStakedAmount[msg.sender];

        _addressToStakedAmount[msg.sender] = 0;
        _stakedAmount -= amount;

        if (_isValidator(msg.sender)) {
            _deleteFromValidators(msg.sender);
        }

        // @audit use call
        payable(msg.sender).transfer(amount);
        emit Unstaked(msg.sender, amount);
    }
```

Description

transfer() is used for native ETH withdrawal.

The transfer() and send() functions forward a fixed amount of 2300 gas. Historically, it has often been recommended to use these functions for value transfers to guard against reentrancy attacks. However, the gas cost of EVM instructions may change significantly during hard forks which may break already deployed contract systems that make fixed assumptions about gas costs. For example. EIP 1884 broke several existing smart contracts due to a cost increase of the SLOAD instruction.

Impact

The use of the deprecated transfer() function for an address will inevitably make the transaction fail when:

- The claimer smart contract does not implement a payable function.
- The claimer smart contract does implement a payable fallback which uses more than 2300 gas unit.
- The claimer smart contract implements a payable fallback function that needs less than 2300 gas units but is called through proxy, raising the call's gas usage above 2300.

Additionally, using higher than 2300 gas might be mandatory for some multisig wallets.

Recommendation

Use call() instead of transfer()

Status

Resolved

Informational Issues

2. Pragma version not locked.

Description

The pragma version is currently set to ^0.8.7, which allows for any minor version above 0.8.7 to be used for compilation. This can introduce unexpected behavior due to changes in the compiler.

Recommendation

Lock the pragma version to 0.8.23 to ensure consistent behavior across compilations

Status

Resolved

Functional Tests

Some of the tests performed are mentioned below:

- ✓ test_IfBLSKeyCanBeChanged()
- ✓ testFail_DeleteValidatorsIfMinimumIsReached()
- ✓ test_DeleteValidators()
- ✓ test_staking()
- ✓ test_unstaking()
- ✓ test_viewFunctions()
- ✓ test_NonValidatorFunctionFlow()
- ✓ test_IfAmountIsLessThenShoudIntBecomeValidator()

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 W Chain. We performed our audit according to the procedure described above.

We Audited the Contract Throughly and Found Only One Low and One Informational severity Issue. In The End,W Chain team Fixed It.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in W Chain. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of W Chain. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of W Chain Team to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we’ve secured over 1000 projects globally, averting over $30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project’s security with cutting-edge practices.

1000+
Audits Completed

$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
December, 2024

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com