QuillAudits

Audit Report
December, 2023

For

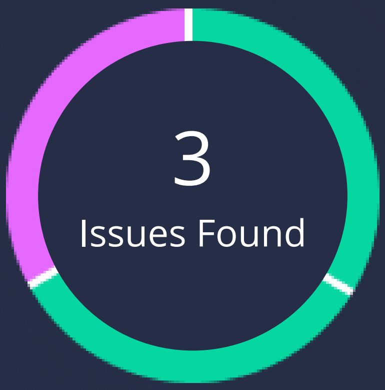
BlockSafe

Table of Content

Executive Summary

Project Name	BlockSafe
Project URL	*https://etherscan.io/address/0x160dbe5c43e98e7897a54451545ba46111a8550c#code*
Overview	A staking contract for users to stake and unstake their $WAIT token in a fixed period of time.
Audit Scope	WaitStaking.sol
Contracts in Scope	WaitStaking.sol
Commit Hash	NA
Language	Solidity
Blockchain	Ethereum
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	December 19, 2023 - December 29, 2023
Updated Code Received	NA
Review 2	NA
Fixed In	NA

Number of Security Issues per Severity



🟥 High 🟨 Medium

🟩 Low 🟪 Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	2	1
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	0	0

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 impacts 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.

A. WaitStaking

High Severity Issues

No issues were found.

Medium Severity Issues

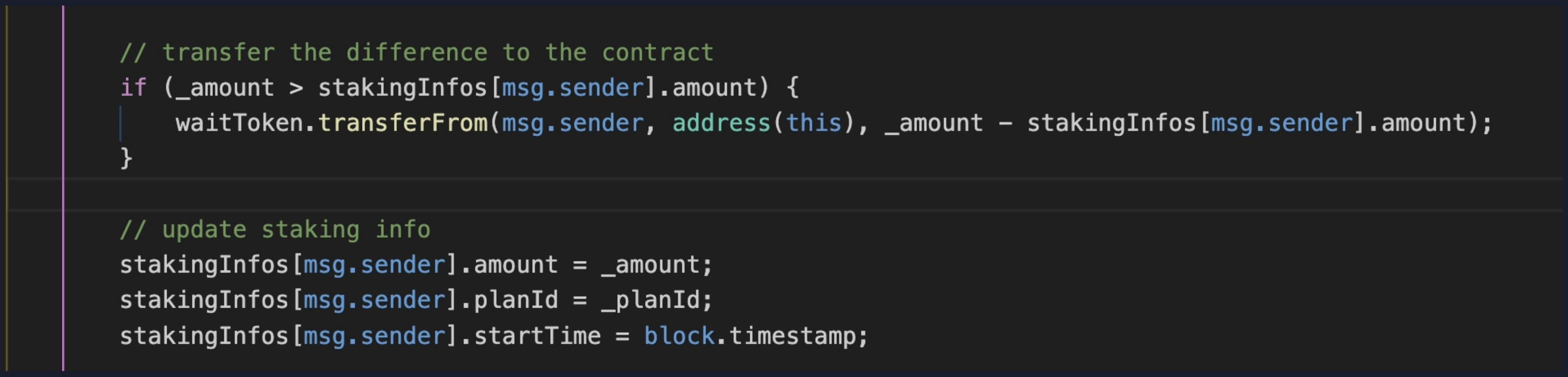
No issues were found.

Low Severity Issues

A.1: Rewards are reset without any record on-chain

Description

As per the contract architecture, rewards are accumulated in the contract for the whole staking period and stored in `accumulatedRewards` at a constant rate. When the stake period ends, the tokens staked are sent back to the user and the reward details are reset. (Similar flow in restake function). This leaves reward details untracked in the contract as well as on block chain as it is not emitted anywhere.



```
// transfer the difference to the contract
if (_amount > stakingInfos[msg.sender].amount) {
    waitToken.transferFrom(msg.sender, address(this), _amount - stakingInfos[msg.sender].amount);
}

// update staking info
stakingInfos[msg.sender].amount = _amount;
stakingInfos[msg.sender].planId = _planId;
stakingInfos[msg.sender].startTime = block.timestamp;
```

Remediation

It is recommended to emit an event to help user track the rewards.

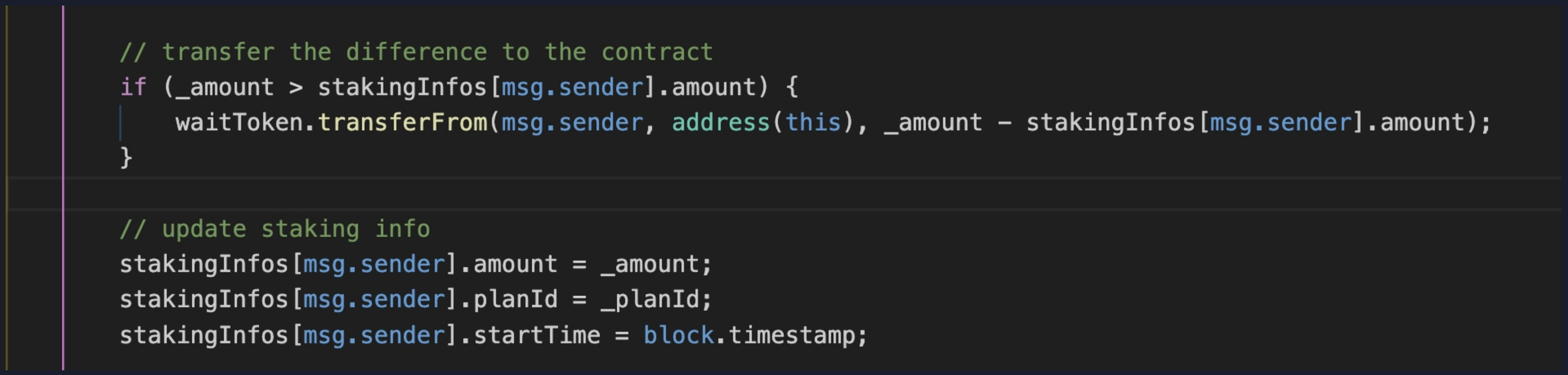
Status

Acknowledged

A.2: Check-effect-interaction pattern

Description

The function stake() uses a transferFrom call before updating the states which violates solidity's recommended check-effect-interaction pattern, as shown below. As wait token is not in scope, it is assumed as an external call and hence it should be called at the end of function.



```
// transfer the difference to the contract
if (_amount > stakingInfos[msg.sender].amount) {
    waitToken.transferFrom(msg.sender, address(this), _amount - stakingInfos[msg.sender].amount);
}

// update staking info
stakingInfos[msg.sender].amount = _amount;
stakingInfos[msg.sender].planId = _planId;
stakingInfos[msg.sender].startTime = block.timestamp;
```

Remediation

Call transfer from after updating stakingInfos states.

Status

Acknowledged

Informational Issues

A.3: Custom Errors

Description

The contract uses require statement for validating states, it is recommended to use custom errors instead to reduce gas costs for returning long strings. They are usually cheaper than string reverts.

Remediation

Use custom errors where possible.

Status

Acknowledged

Functional Tests

Some of the tests performed are mentioned below:

- ✓ [PASS] test stake with token transfer
- ✓ [PASS] test unstake after stake period ends
- ✓ [PASS] test unstake after freezing period ends
- ✓ [PASS] test revert when unstaked before stake period ends
- ✓ [PASS] test revert when unstaked with 0 amount
- ✓ [PASS] test stake with amounts out of range

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 BlockSafe. We performed our audit according to the procedure described above.

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

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in BlockSafe smart contracts. 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 BlockSafe smart contracts. 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 BlockSafe to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

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.

850+
Audits Completed

$30B
Secured

$30B
Lines of Code Audited

Follow Our Journey

Audit Report
December, 2023

For

BlockSafe

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com