QuillAudits

Audit Report
January 2023

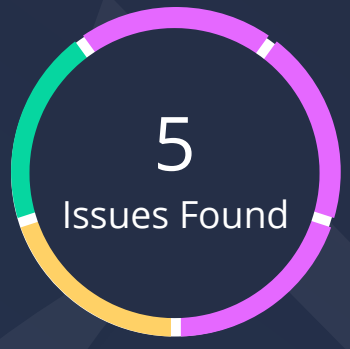
For

d o r a d u s

Table of Content

Executive Summary

Project Name	Doradus
Overview	Doradus contract allows for users to stake in order to yield a substantial reward in the future. The contract inherits three basic contracts from the standard Openzeppelin libraries; Ownable, Pausable, and ReentrancyGuard. These libraries aid in moderation of access control for some functions, pausing of some activities in contract, and also providing guard against reentrancy attacks.
Timeline	14 January, 2023 to 16 January, 2023
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyse Doradus codebase for quality, security, and correctness. *https://bscscan.com/address/0x9e3f4426dbf149ce9c9de84b8cf408ec873da0e7*



High Medium

Low Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	1	1	3
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	0	0

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 - Doradus

High Severity Issues

No issues found

Medium Severity Issues

1. Centralization Related Risks

Description

In contract, there were a couple of critical permissions given to the owner of the contract. One, activities that allow users engagement of the contract is impossible until the contract owner calls the start function. Two, the owner of the contract can moderate when users are to withdraw their funds with the initialize() and deintialize() functions. Third, the contract owner can remove users with the leaveDoradus() function and deny them withdrawal abilities. Lastly, the owner of the contract can withdraw all of the funds in the contract with the doradusPool() function. This excessive permission granted to the contract owner could cause stealing of funds when a malicious owner becomes the new owner of the contract.

Remediation

implement a model of information passage to users before these critical actions are carried out. Some of the functions that grant magnitude permissions to the contract owner can be better achieved with using a multi-sig of trusted persons. This is to get the trust of users that owners of the contract cannot just remove funds from the contract without their awareness.

Status

Acknowledged

Low Severity Issues

2. Missing Zero Address Validation

```
function deposit(address partner⬆) external payable whenStarted nonReentrant {
    require(msg.value >= 10000000000000000, "Too low amount for deposit");
    _updateNotWithdrawn(_msgSender());
    stake[_msgSender()].stake += msg.value;
    if (stake[_msgSender()].percentage == 0) {
        require(partner⬆ != _msgSender(), "Cannot set your own address as partner");
        stake[_msgSender()].partner = partner⬆;
    }
    _updatePercentage(_msgSender());
    emit StakeChanged(_msgSender(), stake[_msgSender()].partner, stake[_msgSender()].
    stake);
}
```

Description

Contract permits the addition of a partner when a user first interacts with the deposit function. However, the deposit function misses an input validation to confirm if the null address was passed into the function. When users pass null addresses unknowingly, all benefits maintained for the partner are irredeemable

Remediation

add sufficient input validation to ensure that users' partners are not set to the null address.

Status

Acknowledged

Informational Issues

3. Unlocked pragma (pragma solidity ^0.8.15)

Description

The basic contract has a floating solidity version. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Remediation

It is recommended to use a locked solidity pragma version to prevent the possibility of a bug introduced by newer versions. Inherited contracts that have been audited could be maintained.

Status

Acknowledged

4. Missing Event Emission

Description

Event emission are critical actions in smart contracts to help track its data on-chain. When this is missing in contracts, it becomes difficult to track some critical state changes happening within the contract. In this contract, it fails to emit events on the following functions:

- withdraw
- leaveDoradus
- start
- doradusPool

Remediation

emit events for these critical functions in order to make the processing of data filtering and tracking possible and easy.

Status

Resolved

5. Gas Optimization (Comparison of Value to Zero)

Description

While this is an important check to do in contract in order to ensure that users are not calling these functions with zero amount value, its model of comparison and computation requires more gas. Carrying out an operation to know if an unsigned integer parameter is greater than 0 is costlier. This is present in two functions: reinvest and withdraw.

Remediation

It is cheaper when the contract ensures that the incoming parameter is not equal to zero value. This is done this way; require(amount != 0, “Zero amount”).

Status

Acknowledged

Functional Testing

Some of the tests performed are mentioned below:

- Should revert when non-owner calls the start function
- Should be able to start contract activities after the contract owner successfully calls start function
- Should revert when users call deposit when contract has not started
- Should revert when users attempt to deposit less than the required stake amount
- Should revert when users set themselves as their partners
- Should be able to deposit successfully and continuously.
- Should revert if user attempts to reinvest with zero amount
- Should revert if user attempts to reinvest when they have not yielded enough
- Should be able to reinvest when yielded funds in contract is enough
- Should revert when users call withdraw but contract is paused
- Should be able to withdraw funds when yielded rewards is substantial
- Should be able to remove users by contract owner when leaveDoradus function is called
- Owner should be able to withdraw funds from the contracts without any hindrance.

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

Some issues of Medium, Low and Informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture. In the End, Doradus Team Acknowledged all Issues.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Doradus 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 Doradus 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

$16B
Secured

700K
Lines of Code Audited

Follow Our Journey

Audit Report
January, 2023

For

doradus

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com