QuillAudits

Audit Report
October, 2023

For

VAULTY

Table of Content

Table of Content

Executive Summary

Project Name	Vaulty
Overview	The GoalVault contract allows users to create various vaults targeted towards meeting a specific goal amount. Vaults are either designed to permit early withdrawal or not. While the TimeVault is designed to help users hold their funds for a period of time with the end goal of earning a reward. Vaults with early withdrawal features attract a penalty fee that could vary.
Method	Manual Review, Automated Testing, Functional Testing, etc.
Language	Solidity
Blockchain	EVM Compatible
Audit Scope	The scope of this audit was to analyze the Vaulty codebase for quality, security, and correctness: *https://github.com/ashwwwin/VaultyGoalVault* https://github.com/ashwwwin/VaultyTimeVault
Branch	Main
Commit	e6d0c283a7cc57154ff27afe59020eea8c3a08f0 1b2fc140ab604b9285a2f3c14816a5e484b885d2
Contracts in Scope	VaultyGoalVault VaultyTimeVault
Initial Audit	20th June 2023 - 30th June 2023
Second Review	20th September 2023
Fixed In	731498d8fde817a01ccb9ac5db3661344ecbc595 d0cf113b0ad05b48d271323ae6ac18b98480bbc4

Number of Security Issues per Severity

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	2	2
Partially Resolved Issues	0	0	0	0
Resolved Issues	1	0	0	4

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- 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 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 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, Truffle, 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.

A. Common Issues

High Severity Issues

No issues were found.

Medium Severity Issues

No issues were found.

Low Severity Issues

A.1 Missing Event Emission for Significant Actions

Description

Whenever certain significant actions are performed within the contract, it is recommended to emit an event about it. Significant actions such as vault creation, setting the creation fee and updating the whitelist manager. While it is possible to track emitted events made from token contracts when depositing and withdrawing from vaults, event emissions are important to protocol to help track state changes. These functions to emit for are as follows:

- createVault/createGoal
- setCreationFees
- setWhitelistManager

Remediation

Consider emitting an event whenever certain significant changes are made in the contracts.

Status

Acknowledged

A.2 Missing Check for Zero Address

Description

There are critical functions that allow for the owner to set a whitelist manager and for users to transfer their vaults but these functions fail to check that the addresses being passed as parameters is not a null address. This is also an issue with creating vaults. When users make the null address, it is impossible to claim such vaults afterwards.

A.2 Missing Check for Zero Address

This is a common issue among both contracts in scope. Affected functions are as follows:

- createVault/createGoal
- setWhitelistManager
- transferVault

Remediation

Add a null address check to these functions to prevent setting a null address.

Status

Acknowledged

Informational Issues

A.3 Absence of Proper Code Comment in All Contracts

Description

Proper code comments explain the purpose of functions in the contracts and also about the kind of parameters a function expects as an input value. This is why it is recommended that contracts should have a NATSPEC code format to give a comprehensive meaning to the functions. While there are few comments that explain the data type and few lines of the code, it is not substantial.

Remediation

Add a proper natspec comment format across the contracts.

Status

Acknowledged

Reference

https://docs.soliditylang.org/en/v0.8.20/style-guide.html#natspec

A.4 Prevent Gas Wastage on Transfer of Zero Value for Some Tokens

Description

Some ERC20 tokens are designed to revert when called transferring zero values. There are some tokens that exempt this check and when they get whitelisted, it is possible for the contract owner to call the withdrawPenaltyFee function for these tokens and this won't

A.4 Prevent Gas Wastage on Transfer of Zero Value for Some Tokens

revert, causing a wastage of gas after the call.

Remediation

Adding a check will help know when the contract balance of that token exceeds zero and afterwards transfer that token amount to the contract owner.

Status

Resolved

A.5 Functions Certain to Revert When Called by Normal Users can be Marked Payable

Description

With the use of modifiers on some functions restrict the privilege of who can call them. This gives us the certainty of some functions reverting when called by users not allowed to call this function, it is recommended to mark these functions as payable as this will save gas when they are called by privileged addresses.

Remediation

Functions with the onlyOwner functions should be marked payable to save gas.

Status

Resolved

A.6 Use != 0 to Minimize Gas Instead of >0 for Unsigned Comparison

Description

For gas optimization purposes, there are some unsigned comparisons made in both contracts to ensure that a function is greater than zero. This is recommended to use a check that ensures that a value is a non-zero value to save some gas rather than it being exactly greater than zero.

Remediation

Use != 0 rather than >0.

Status

Resolved

B. VaultyGoalVault.sol

High Severity Issues

No issues were found.

Medium Severity Issues

No issues were found.

Low Severity Issues

No issues were found.

Informational Issues

B.1 Incorrect Check in TransferVault Function

```
function transferVault(uint256 vaultId, address newOwner) external {
    Vault storage userGoal = vaultInfo[vaultId];

    require(userGoal.owner == msg.sender, "Only the creator can transfer the vault.");
    require(userGoal.balance == 0, "Cannot transfer vault with tokens in it.");

    userGoal.owner = newOwner;
}
```

Description

Vaults are designed to be associated with owners and the system allows for these vaults to be transferred between different addresses. However, for vaults to be transferred, they are expected to have a non-zero balance. In the transferVault function, the check is otherwise; expecting that a vault must be zero balance before it can be transferred.

Remediation

Consider correcting the check in the function.

Status

Resolved

B.2 Do Not Initialize Variables With Default Values

```
uint256 public creationFee = 0;
uint256 public penaltyPercent = 0;
uint256 public _vaultIdCounter = 0;

uint256 private creationFeePool = 0;
```

Description

State variables that are certain to be updated when called by several functions in the codebase should not be initialized and set to default values in order to save gas.

Remediation

Initialize these state variables without setting them to zero.

Status

Acknowledged

C. VaultyTimeVault.sol

High Severity Issues

C.1 Qualified Users May Lose Rewards when They Call Withdraw Function First

Description

To be qualified to earn rewards as a user, a vault must meet some requirement they get from the portion of the reward pool. Of course, the user calling the vault of which to claim reward must be the owner, other requirements are that users must have some tokens in the vault, must not have claimed the rewards before, and that the creation time for the vault must be lesser than the timestamps allotted for the rewards. The issue arises when users call the withdraw function first before calling the claimRewards function. This is because at the point of claiming rewards, users amount will already be made to zero, denying them the right to claim rewards even though they met other requirements before calling the withdraw function.

Remediation

Redesign the system of withdrawal to sum up the total amount in the vault with the expected rewards for a vault, before the token is transferred to the users.

Status

Resolved

NB

Client chooses to handle the reward calculations and distribution mechanism from another claim contract. In the Vaulty Time contract, all accumulated users reward are forwarded to the reward manager contract which begins the process of reward disbursement on the protocol.

Medium Severity Issues

No issues were found.

Low Severity Issues

No issues were found.

Informational Issues

No issues were found.

Functional Tests

Some of the tests performed are mentioned below:

- Should set and get critical state variables that influences the creation of vaults
- Should revert when non-privileged addresses attempts to call privileged functions
- Should allow the contract owner to set the whitelist manager and whitelisted manager calls some of its privileged functions
- Should allow the creation of vaults by different addresses and ensuring vaults id counter increases
- Should revert when users do not send the exact amount for vault creation fee
- Should revert when users intend to create vaults with non-whitelisted tokens
- Should revert when users create goal vault with the intend goal value lesser than amount to send into the newly created vault
- Should allow users deposit into the goal vaults until it meets or exceed expected goal for a vault
- Should revert when users attempt to withdraw from vaults with early withdrawal deactivated
- Should revert when users tries to withdraw from a vault whose balance has not met its intended vault goal
- Should increase pool of derived penalty fee from withdrawal made from vaults with early withdrawal feature
- Should only transfer vaults when it is called by the owner and the vault has some balance
- Should successfully withdraw ethers derived from creation of vaults
- Should allow only owner begin ownership transfer and pending owner complete the processes
- Should revert when ownership transfer processes is not completed within 5 seconds
- Should claim rewards for users with vaults created before rewards pool are accumulated
- Should reverts when new vaults not qualified for rewards attempts to claim rewards

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.

```
INFO:Detectors:
Reentrancy in VaultyGoalVault.deposit(uint256,uint256) (src/VaultyGoalVault.sol#70-78):
        External calls:
        - IERC20(userGoal.token).safeTransferFrom(msg.sender,address(this),amount) (src/VaultyGoalVault.sol#74)
        State variables written after the call(s):
        - userGoal.balance += amount (src/VaultyGoalVault.sol#77)
        VaultyGoalVault.vaultInfo (src/VaultyGoalVault.sol#40) can be used in cross function reentrancies:
        - VaultyGoalVault.createGoal(address,uint256,uint256,address,bool) (src/VaultyGoalVault.sol#42-68)
        - VaultyGoalVault.deposit(uint256,uint256) (src/VaultyGoalVault.sol#70-78)
        - VaultyGoalVault.earlyWithdraw(uint256) (src/VaultyGoalVault.sol#92-109)
        - VaultyGoalVault.transferVault(uint256,address) (src/VaultyGoalVault.sol#111-118)
        - VaultyGoalVault.vaultInfo (src/VaultyGoalVault.sol#40)
        - VaultyGoalVault.withdraw(uint256) (src/VaultyGoalVault.sol#80-90)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
VaultyGoalVault.createGoal(address,uint256,uint256,address,bool).owner (src/VaultyGoalVault.sol#42) shadows:
        - Ownable.owner() (lib/openzeppelin-contracts/contracts/access/Ownable.sol#43-45) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
VaultyGoalVault.setCreationFee(uint256) (src/VaultyGoalVault.sol#166-168) should emit an event for:
        - creationFee = fee (src/VaultyGoalVault.sol#167)
VaultyGoalVault.setPenaltyPercent(uint256) (src/VaultyGoalVault.sol#170-173) should emit an event for:
        - penaltyPercent = percent (src/VaultyGoalVault.sol#172)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
VaultyGoalVault.setWhitelistManager(address).manager (src/VaultyGoalVault.sol#135) lacks a zero-check on :
                - whitelistManager = manager (src/VaultyGoalVault.sol#136)
VaultyGoalVault.startOwnershipTransfer(address).newOwner (src/VaultyGoalVault.sol#186) lacks a zero-check on :
                - pendingOwner = newOwner (src/VaultyGoalVault.sol#187)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in VaultyGoalVault.createGoal(address,uint256,uint256,address,bool) (src/VaultyGoalVault.sol#42-68):
        External calls:
        - IERC20(token).safeTransferFrom(msg.sender,address(this),amount) (src/VaultyGoalVault.sol#49)
        State variables written after the call(s):
        - _vaultIdCounter ++ (src/VaultyGoalVault.sol#52)
        - creationFeePool += msg.value (src/VaultyGoalVault.sol#67)
        - userGoal.vaultId = vaultId (src/VaultyGoalVault.sol#57)
        - userGoal.token = token (src/VaultyGoalVault.sol#58)
        - userGoal.goal = goal (src/VaultyGoalVault.sol#59)
        - userGoal.balance = amount (src/VaultyGoalVault.sol#60)
        - userGoal.isWithdrawableEarly = isWithdrawableEarly (src/VaultyGoalVault.sol#61)
        - userGoal.penaltyPercent = penaltyPercent (src/VaultyGoalVault.sol#62)
        - userGoal.owner = owner (src/VaultyGoalVault.sol#63)
        - userGoal.createdBy = msg.sender (src/VaultyGoalVault.sol#64)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
VaultyGoalVault.completeOwnershipTransfer() (src/VaultyGoalVault.sol#191-199) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp < ownerChangeTimeout,Ownership transfer has timed out.) (src/VaultyGoalVault.sol#193)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address._revert(bytes,string) (lib/openzeppelin-contracts/contracts/utils/Address.sol#231-243) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts/contracts/utils/Address.sol#236-239)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Different versions of Solidity are used:
        - Version used: ['0.8.1', '^0.8.0', '^0.8.1']
        - 0.8.1 (src/VaultyGoalVault.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/access/Ownable.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/IERC20.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/extensions/IERC20Permit.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/utils/Context.sol#4)
        - ^0.8.1 (lib/openzeppelin-contracts/contracts/utils/Address.sol#4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Pragma version^0.8.0 (lib/openzeppelin-contracts/contracts/access/Ownable.sol#4) allows old versions
Pragma version^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/IERC20.sol#4) allows old versions
Pragma version^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/extensions/IERC20Permit.sol#4) allows old versions
Pragma version^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol#4) allows old versions
Pragma version^0.8.1 (lib/openzeppelin-contracts/contracts/utils/Address.sol#4) allows old versions
Pragma version^0.8.0 (lib/openzeppelin-contracts/contracts/utils/Context.sol#4) allows old versions
Pragma version0.8.1 (src/VaultyGoalVault.sol#4) allows old versions
solc-0.8.1 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in SafeERC20._callOptionalReturnBool(IERC20,bytes) (lib/openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol#134-142):
        - (success,returndata) = address(token).call(data) (lib/openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol#139)
Low level call in Address.sendValue(address,uint256) (lib/openzeppelin-contracts/contracts/utils/Address.sol#64-69):
        - (success) = recipient.call{value: amount}() (lib/openzeppelin-contracts/contracts/utils/Address.sol#67)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (lib/openzeppelin-contracts/contracts/utils/Address.sol#128-137):
        - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts/contracts/utils/Address.sol#135)
Low level call in Address.functionStaticCall(address,bytes,string) (lib/openzeppelin-contracts/contracts/utils/Address.sol#155-162):
        - (success,returndata) = target.staticcall(data) (lib/openzeppelin-contracts/contracts/utils/Address.sol#160)
Low level call in Address.functionDelegateCall(address,bytes,string) (lib/openzeppelin-contracts/contracts/utils/Address.sol#180-187):
        - (success,returndata) = target.delegatecall(data) (lib/openzeppelin-contracts/contracts/utils/Address.sol#185)
```

```
VaultyTimeVault.createVault(address,uint256,uint256,bool,address).owner (src/VaultyTimeVault.sol#60) shadows:
        - Ownable.owner() (lib/openzeppelin-contracts/contracts/access/Ownable.sol#43-45) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
VaultyTimeVault.setCreationFee(uint256) (src/VaultyTimeVault.sol#258-260) should emit an event for:
        - creationFee = fee (src/VaultyTimeVault.sol#259)
VaultyTimeVault.setPenaltyPercent(uint256) (src/VaultyTimeVault.sol#262-265) should emit an event for:
        - penaltyPercent = percent (src/VaultyTimeVault.sol#264)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
VaultyTimeVault.setWhitelistManager(address).manager (src/VaultyTimeVault.sol#226) lacks a zero-check on :
                - whitelistManager = manager (src/VaultyTimeVault.sol#227)
VaultyTimeVault.startOwnershipTransfer(address).newOwner (src/VaultyTimeVault.sol#278) lacks a zero-check on :
                - pendingOwner = newOwner (src/VaultyTimeVault.sol#279)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in VaultyTimeVault.createVault(address,uint256,uint256,bool,address) (src/VaultyTimeVault.sol#60-90):
        External calls:
        - IERC20(token).safeTransferFrom(msg.sender,address(this),amount) (src/VaultyTimeVault.sol#67)
        State variables written after the call(s):
        - _vaultIdCounter ++ (src/VaultyTimeVault.sol#73)
        - creationFeePool += msg.value (src/VaultyTimeVault.sol#89)
        - totalTokensLocked[token] += amount (src/VaultyTimeVault.sol#70)
        - vaultInfo[vaultId] = Vault(vaultId,token,amount,unlockTime,isWithdrawableEarly,penaltyPercent,owner,block.timestamp) (src/VaultyTimeVault.sol#77-86)
Reentrancy in VaultyTimeVault.earlyWithdraw(uint256) (src/VaultyTimeVault.sol#111-135):
        External calls:
        - IERC20(userDeposit.token).safeTransfer(msg.sender,amountToTransfer) (src/VaultyTimeVault.sol#126)
        State variables written after the call(s):
        - distributeFees(userDeposit.token,fee,timestamp) (src/VaultyTimeVault.sol#134)
                - penaltyFeePool[token] += protocolFee (src/VaultyTimeVault.sol#161)
        - distributeFees(userDeposit.token,fee,timestamp) (src/VaultyTimeVault.sol#134)
                - rewardPerToken[token][timestamp] = rewardsPool[token][timestamp] / totalTokensLocked[token] (src/VaultyTimeVault.sol#167)
        - distributeFees(userDeposit.token,fee,timestamp) (src/VaultyTimeVault.sol#134)
                - rewardsPool[token][timestamp] += rewardsAvailable (src/VaultyTimeVault.sol#164)
        - totalTokensLocked[userDeposit.token] -= amount (src/VaultyTimeVault.sol#129)
Reentrancy in VaultyTimeVault.withdraw(uint256) (src/VaultyTimeVault.sol#93-109):
        External calls:
        - IERC20(userDeposit.token).safeTransfer(msg.sender,amount) (src/VaultyTimeVault.sol#105)
        State variables written after the call(s):
        - totalTokensLocked[userDeposit.token] -= amount (src/VaultyTimeVault.sol#108)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in VaultyTimeVault.earlyWithdraw(uint256) (src/VaultyTimeVault.sol#111-135):
        External calls:
        - IERC20(userDeposit.token).safeTransfer(msg.sender,amountToTransfer) (src/VaultyTimeVault.sol#126)
        Event emitted after the call(s):
        - feesDistributed(token,amount,timestamp) (src/VaultyTimeVault.sol#169)
                - distributeFees(userDeposit.token,fee,timestamp) (src/VaultyTimeVault.sol#134)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
VaultyTimeVault.createVault(address,uint256,uint256,bool,address) (src/VaultyTimeVault.sol#60-90) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(unlockTime > block.timestamp,unlockTime must be in the future) (src/VaultyTimeVault.sol#64)
VaultyTimeVault.withdraw(uint256) (src/VaultyTimeVault.sol#93-109) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= userDeposit.unlockTime,Vault is still locked.) (src/VaultyTimeVault.sol#97)
VaultyTimeVault.completeOwnershipTransfer() (src/VaultyTimeVault.sol#283-291) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp < ownerChangeTimeout,Ownership transfer has timed out.) (src/VaultyTimeVault.sol#285)
```

Closing Summary

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

Some issues of high, 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 Vaulty 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 Vaulty 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.

850+
Audits Completed

$30B
Secured

$30B
Lines of Code Audited

Follow Our Journey

Audit Report
October, 2023

For

VAULTY

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com