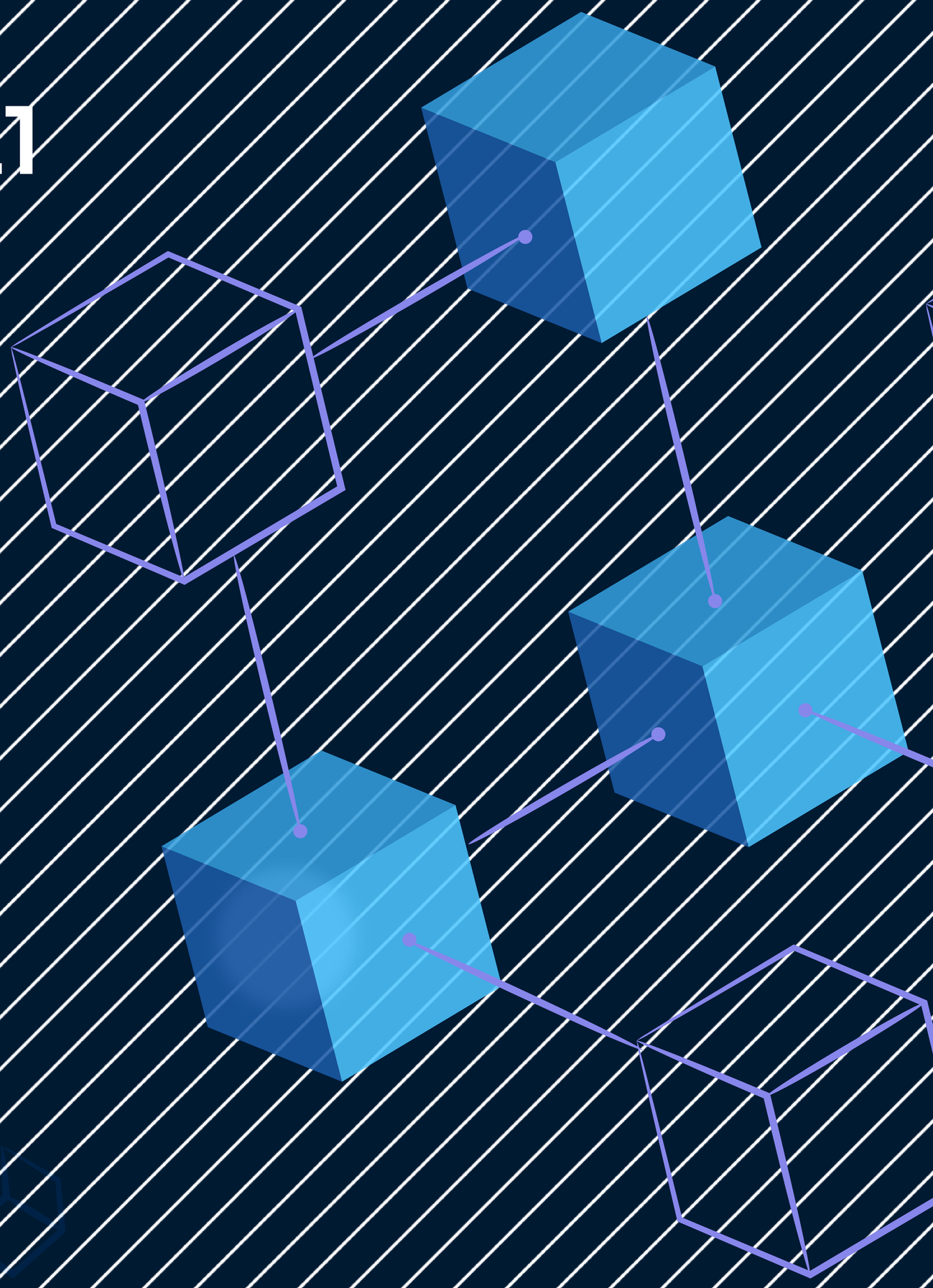
QuillAudits

Audit Report
November, 2021



For

checkdot

Contents

Scope of Audit	01
Check Vulnerabilities	01
Techniques and Methods	02
Issue Categories	03
Number of security issues per severity.	03
Introduction	04
Issues Found – Code Review / Manual Testing	05
High Severity Issues	05
1. Missing Check for max that users can send	05
Medium Severity Issues	06
2. Race condition if pause is not done before claim	06
3. Use of payable in non eth receiving functions	06
4. Floating Pragma Solidity and Latest Version	07
Low Severity Issues	08
5. Code reusability	08
6. Specific value of CDT tokens should be withdrawn	08
Informational Issues	09
7. Specified functions visibility	09
Test Cases for Functional Testing	10
Testnet Testing Transaction Hash	11
Functional Tests	12
Closing Summary	13

Scope of the Audit

The scope of this audit was to analyze and document the Checkdot Token smart contract codebase for quality, security, and correctness.

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- 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 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, Theo.

Issue Categories

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.

Risk-level	Description
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.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	2	2	0
Closed	1	1	0	1

Introduction

During the period of **Oct 25, 2021 to Oct 31, 2021** - QuillAudits Team performed a security audit for **Checkdot** smart contracts.

The code for the audit was taken from the following official link: https://github.com/checkdot/CheckDotPrivateSaleContract/blob/master/contracts/CheckDotPrivateSale.sol

Ver	Date	Commit hash	Note
1	23-10-21	84237ce510704d2761d4b88167686be2ac5ad807	First Code
2	30-10-21	161762cebfd42da0730d580c6a46ec5182e26b5e	Fixed Code

Issues Found

A. Contract – CheckDotPrivateSale

High severity issues

1. Missing Check for max that users can send

Description

Missing Check for max that users can send, multiple times this is not showing error in metamask and user sends the transaction then this may result in transaction failure and gas fees wastage. This can be minimized by using the modifier in the receive() function directly.

Remediation

Require check in the receive() function also.

Line	Code

49	receive() external payable {
	require(_paused == false, "Presale is paused");
	uint256 totalInvested = _wallets_investment[address(msg.sender)].add(msg.value);
	require(totalInvested <= _maxethPerWallet, "You depassed the limit of max eth per wallet for the presale.");
	_transfertCDT(msg.value);
	}

Status: Fixed and Closed

Medium severity issues

2. Renounce Ownership

Description
Race condition if pause is not done before claim, or if claim is set to true before pause is true.

Remediation
Require check for pause and claim in sequential manner to avoid mismatch value and race condition.

Status: Fixed and Closed

Line	Code

59	function setPaused(bool value) public payable onlyOwner {
	_paused = value;
	emit StateChange();
	}

3. Use of payable in non eth receiving functions

Description
Payable is used only when we receive eth in the function. As per the contract description only one function is receiving eth, but actually multiple functions accept ETH, this may lead to mismatch in ETH and tokens values if the owner sends ETH to the contract by owner only ETH functions. Payables should only be used with the functions needed to receive ETH or ETH transacting to any address.

Remediation
To save gas and make code optimised, etra payable functions should be checked and payable should be used only when needed.

Status: Fixed and Closed

Line	Code

67	function setPaused(bool value) public payable onlyOwner {
	_paused = value;
	emit StateChange();
	}
	
	function setClaim(bool value) public payable onlyOwner {
	_claim = value;
	emit StateChange();
	}

4. Floating pragma solidity and latest version

Description

Floating pragma is not recommended as some functions are version dependents and recommended to use strict solidity versions in pragma only. Also the latest version of solidity is not recommended as this may include some experimental features, untracked bugs and time evaluation in the production environment.

Remediation

Switch to fixed pragma, also use a slightly lower version of solidity pragma.

Status: Partially Changed Acknowledged and closed

Line	Code
03	pragma solidity ^0.8.9;

Low level severity issues

5. Code reusability

Description
IERC20 cdtToken is defined multiple times internally in the function which can be defined once in the contract definition and can be reused.

Remediation
Declare IERC20 cdtToken outside the functions in the header of the file and reuse.

Status: Fixed and Closed

Line	Code
157	IERC20 cdtToken = IERC20(_cdtTokenAddress);
77	IERC20 cdtToken = IERC20(_cdtTokenAddress);

6. Specify value of CDT tokens should be withdrawn.

Description
In case if some extra CDT is required to be withdraw after the sale, then it will be difficult to withdraw as this function only withdraws all the CDT at the same time.

Remediation
Recommended to input specific values of CDT and withdraw to maintain the logic and carry out the withdraws operation. So that pending withdraws will not run out of funds

Status: Fixed and Closed

Line	Code

156	function withdrawRemainingCDT() public payable onlyOwner {
	IERC20 cdtToken = IERC20(_cdtTokenAddress);
	
	cdtToken.transfer(msg.sender, cdtToken.balanceOf(address(this)));
	}

Informational Issues

7. Specify functions visibility

Description
Functional visibility should be specific to save gas and limit the usage of the function to optimise the code.

Remediation
Use external where functions are not required to be public.

Status: Partially Fixed and Closed

Line	Code
59	function setPaused(bool value) public payable onlyOwner
67	function setClaim(bool value) public payable onlyOwner {
95	function getMaxEthPerWallet() public view
102	function getCdtPerEth() public view returns(uint256)
109	function getTotalRaisedEth() public view returns(uint256) {
	And other public functions in the contract to be checked....

Test Cases for Functional Testing

Contract : CheckDotPrivateSale.sol

- Test library functions and versions. [**PASSED**]
- Validation of library solidity version. **[PASSED]**
- Check the compatibility of the library function with the used solidity version. **[PASSED]**
- Check for solidity version used. **[Acknowledged]**
- Solidity version and code contents compatibility. **[PASSED]**
- Compilability and byte code verification. **[PASSED]**
- Standard function use and correctness. **[PASSED]**
- Owner functions and validation. **[PASSED]**
- Owner address checks, risks, availability, updateable, risks, and use case compatible. **[PASSED]**
- Functions visibility and use. **[Solved and Acknowledged]**
- Modifier types and reusability. **[PASSED]**
- Code optimization and validations for every function. **[PASSED]**
- Check for deprecated functionality. **[PASSED]**
- Check for overflow, underflow values, and calculations. **[PASSED]**
- Check for incoming and outgoing values. **[PASSED]**
- Checks for missing checks on values and possible attacks. **[PASSED]**
- Reentering guard and checking for race condition. **[PASSED]**
- Checks for multiple hitting a function at the same time in case of race condition. **[PASSED]**
- Visibility and missing checks for users. **[PASSED]**
- Check the use case and compare it with the implementation of the function. **[PASSED]**
- Check for time and timestamp validation and correctness. **[PASSED]**
- Decimal values accuracy and value matching. **[PASSED]**

- Checks for user possible attack conditions in case of missing checks. **[PASSED]**
- ETH security for withdrawing function, validation for ETH. **[PASSED]**
- Check for formula and arithmetic operations for the contract logic. **[PASSED]**
- Checks for floating values and dependencies from packages. **[PASSED]**
- Check for block and msg dependent values. **[PASSED]**
- Check for open user functions and validations. **[PASSED]**
- Check for Eherscan value display and correctness. **[PASSED]**
- Block values and dependent values checks. **[PASSED]**
- User’s token security and validation before withdrawing. **[PASSED]**
- Secure withdrawal and check for withdrawal values and display correct messages. **[PASSED]**
- Check for commenting and code description for user transparency. **[PASSED]**

Testnet Testing Transaction Hash

https://ropsten.etherscan.io/address/0x207c926c3221836d4776e8ac45248f2e688ff0f2

https://ropsten.etherscan.io/address/0x7ead2e129a571799c201155195630ba1dbfd2c5d

Functional test

Function Names	Testing results
setPaused()	Passed
setClaim()	Passed
claimCdt()	Passed
getCdtPerEth()	Passed
getTotalRaisedEth()	Passed
getTotalRaisedCdt()	Passed
getAddressInvestment()	Passed
_transfertCDT()	Passed
withdraw	Passed
withdrawRemainingCDT	Passed

Closing Summary

Multiple issues are found under medium, low and information issues. Recommended to check and resolve the issues before actual deployment for best code practice and bug-free code. This report is generated after multiple test cases and the scenarios for attack, values overflow, underflow and multiple security factors under multiple conditions.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or endorsement of the **Checkdot** 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 **Checkdot** Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report
November, 2021

For

checkdot

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com