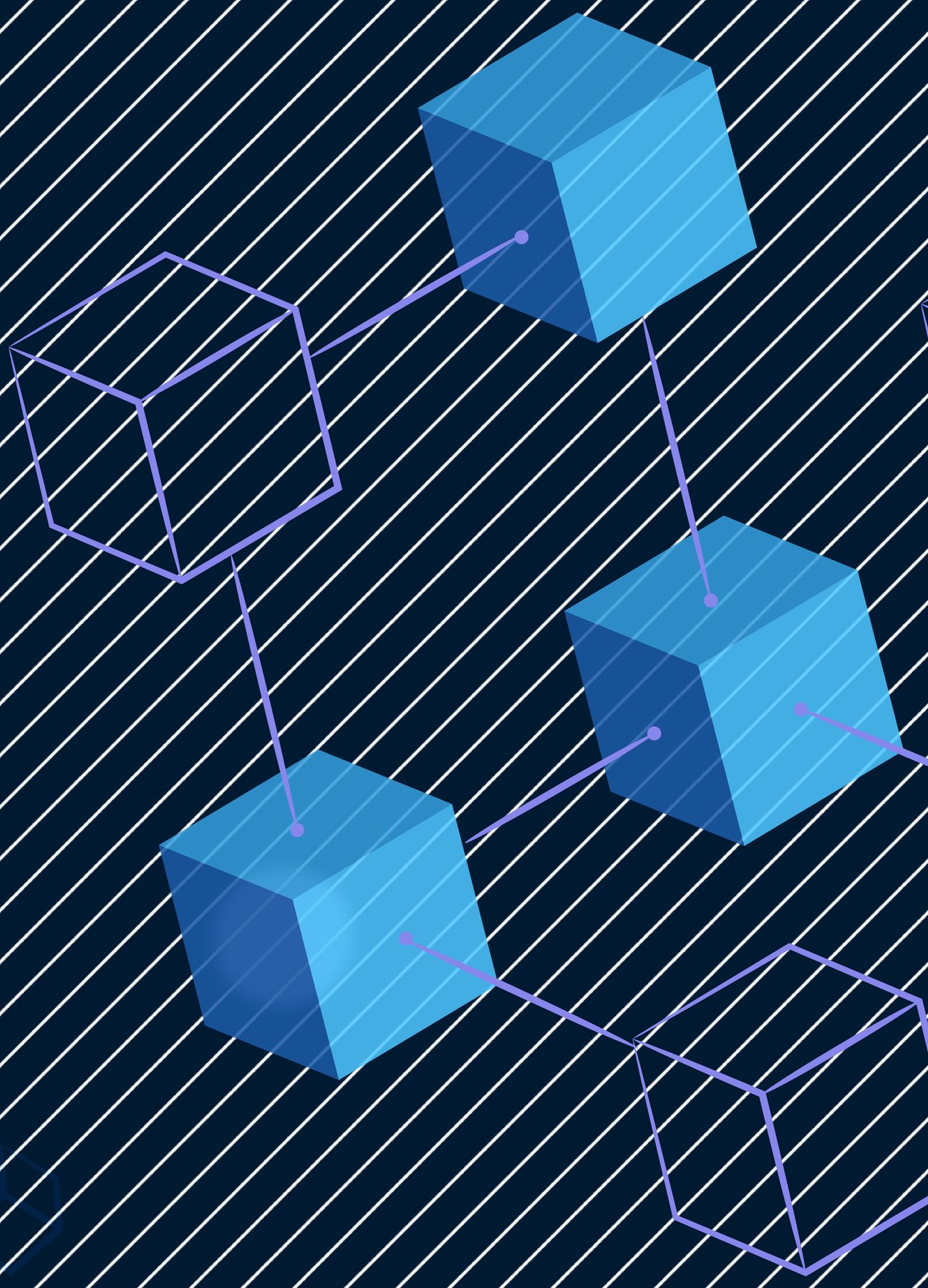
QuillAudits

Audit Report
October, 2021



For

checkdot

Contents

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/BEP-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, Solhint, Slither, Solidity statistic analysis.

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	0	1	0
Closed	0	0	1	0

Introduction

During the period of **October 15, 2021 to October 18, 2021** - QuillAudits Team performed a security audit for Checkdot smart contracts.

The code for the audit was taken from following the official link: https://github.com/checkdot/CheckdotERC20Contract

Note	Date	Commit hash
Version 1	October 15	8a47ef13ace93e0bd473fb1c3893790569a2adc8
Version 2	October 18	1f68a6acd49ae28655fae5882384deb69afd2a7d

Issues Found

A. Contract – Checkdot

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

1. **Lack of Zero address validation**

   Description
   To favor explicitness, consider adding a check for all functions that are taking address parameters in the entire codebase. Although most of the functions throughout the codebase properly validate function inputs, there are some instances of functions that do not. One example is:
   - constructor - does not check for zero address

   Remediation
   Consider implementing require statements where appropriate to validate all user-controlled input, including governance functions, to avoid the potential for erroneous values to result in unexpected behaviors or wasted gas.

   Status: Fixed in version 02

2. **Using old Solidity version**

   Throughout the contract, Solidity 0.8.0 is used and is deployable. However, at the time of writing there are 2 known bugs in version 0.8.0. Consider upgrading your contracts to use the latest version of Solidity, 0.8.9.

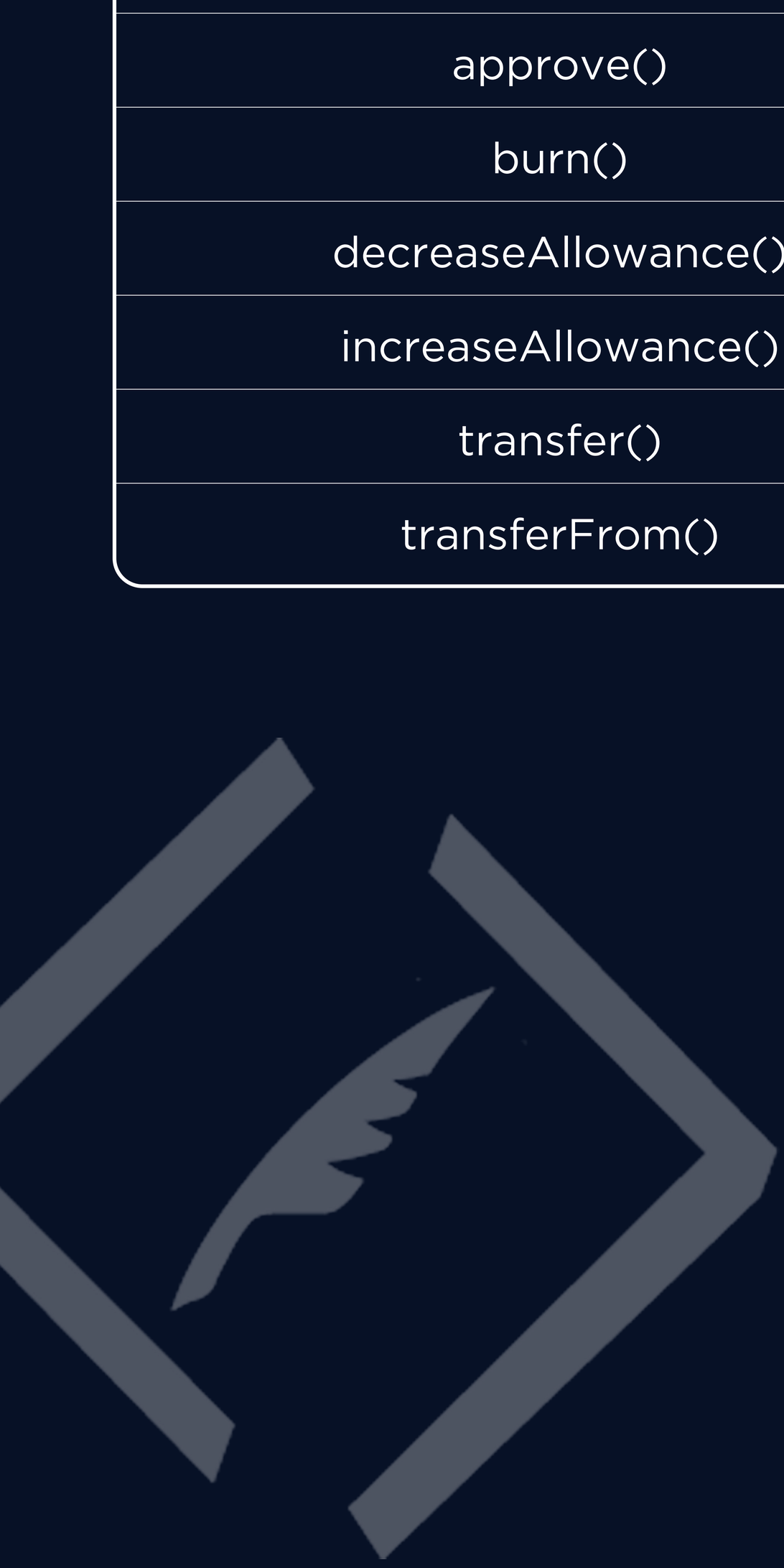
   Status: Acknowledged by the Auditee
   However, the contract does not use the known bugs on 0.8.0 (SignedImmutables, ABIDecodeTwoDimensionalArrayMemory, KeyccakCaching)

Informational issues

No issues were found.

Functional tests

Function Names	Testing results
approve()	Passed
burn()	Passed
decreaseAllowance()	Passed
increaseAllowance()	Passed
transfer()	Passed
transferFrom()	Passed



Automated Tests

Slither

```
INFO:Detectors:
CheckDot._totalSupply (CheckDot.sol#11) shadows:
        - ERC20._totalSupply (erc20/ERC20.sol#37)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing
INFO:Detectors:
CheckDot.constructor(address)._deployer (CheckDot.sol#15) lacks a zero-check on :
                - _checkDotDeployer = _deployer (CheckDot.sol#16)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Context._msgData() (erc20/Context.sol#20-23) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.0 (CheckDot.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (erc20/Context.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (erc20/ERC20.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (erc20/interfaces/IERC20.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.3 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Constant CheckDot._decimals (CheckDot.sol#9) is not in UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (erc20/Context.sol#21)" inContext (erc20/Context.sol#15-25)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
CheckDot.slitherConstructorVariables() (CheckDot.sol#7-25) uses literals with too many digits:
        - _totalSupply = 10000000 * (10 ** uint256(_decimals)) (CheckDot.sol#11)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
CheckDot._totalSupply (CheckDot.sol#11) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
```

```
INFO:Detectors:
name() should be declared external:
        - ERC20.name() (erc20/ERC20.sol#59-61)
symbol() should be declared external:
        - ERC20.symbol() (erc20/ERC20.sol#67-69)
decimals() should be declared external:
        - ERC20.decimals() (erc20/ERC20.sol#84-86)
totalSupply() should be declared external:
        - ERC20.totalSupply() (erc20/ERC20.sol#91-93)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (erc20/ERC20.sol#98-100)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (erc20/ERC20.sol#110-113)
allowance(address,address) should be declared external:
        - ERC20.allowance(address,address) (erc20/ERC20.sol#118-120)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (erc20/ERC20.sol#129-132)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (erc20/ERC20.sol#147-155)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (erc20/ERC20.sol#169-172)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (erc20/ERC20.sol#188-194)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

Mythril

```
==== Potential denial-of-service if block gas limit is reached ====
SWC ID: 128
Severity: Low
Contract: 0xe422E5a167D33cFff7aC6FfD96Fb22aEb2dB65cd
Function name: gasprice_bit_ether(int128)
PC address: 4705
Estimated Gas Usage: 1858 - 2333
Potential denial-of-service if block gas limit is reached.
A storage modification is executed in a loop. Be aware that the transaction may fail to execute if the loop is unbounded and the necessary gas exceeds the block gas limit.
--------------------
Initial State:

Account: [ATTACKER], balance: 0x1, nonce:0, storage:{}
Account: [SOMEGUY], balance: 0x2400021300, nonce:0, storage:{}

Transaction Sequence:

Caller: [SOMEGUY], function: approve(address,uint256), txdata: 0x095ea7b3000000000000000000000000010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010
1010101010101, value: 0x0
Caller: [ATTACKER], function: gasprice_bit_ether(int128), txdata: 0x23b872dd0000000000000000000000000101010101010101010101010101010101010101000000000000000000000000010101010101010101010101
01010101010101010000000000000000000000000000000000000000000000000000000000000000, value: 0x0

==== Potential denial-of-service if block gas limit is reached ====
SWC ID: 128
Severity: Low
Contract: 0xe422E5a167D33cFff7aC6FfD96Fb22aEb2dB65cd
Function name: increaseAllowance(address,uint256)
PC address: 4757
Estimated Gas Usage: 1775 - 2630
Potential denial-of-service if block gas limit is reached.
A storage modification is executed in a loop. Be aware that the transaction may fail to execute if the loop is unbounded and the necessary gas exceeds the block gas limit.
--------------------
Initial State:

Account: [ATTACKER], balance: 0x1, nonce:0, storage:{}
Account: [SOMEGUY], balance: 0x100000000000161, nonce:0, storage:{}

Transaction Sequence:

Caller: [SOMEGUY], function: approve(address,uint256), txdata: 0x095ea7b3000000000000000000000000010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010
1010101010101, value: 0x0
Caller: [ATTACKER], function: increaseAllowance(address,uint256), txdata: 0x3950935100000000000000000000000001010101010101010101010101010101010101010101010101010101010101010101010101010101
010101010101010101010101, value: 0x0
```

SOLHINT LINTER

```
CheckDot.sol
   3:1   error    Compiler version ^0.8.0 does not satisfy the ^0.5.8 semver requirement              compiler-version
   9:5   warning  Constant name must be in capitalized SNAKE_CASE                                     const-name-snakecase
  15:5   warning  Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)  func-visibility

✖ 3 problems (1 error, 2 warnings)
```

Results

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 Checkdot platform. We performed our audit according to the procedure described above.

The audit showed several low severity issues. In the end, the issues were fixed or acknowledged by the Auditee.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an 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
October, 2021

For

checkdot

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com