Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: VeChain Token
Website: vechain.org
Platform: Ethereum
Language: Solidity
Date: April 8th, 2024

Table of contents

Email: audit@EtherAuthority.io

`

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Email: audit@EtherAuthority.io

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the smart contracts of VeChain Token from vechain.org were audited. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on April 8th, 2024.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

- The VEN token contract adheres to the ERC20 standard.
- The VENSale contract is utilized for the sale and distribution of VEN tokens.
- The VeChain (VEN) Token is a standard smart contract with functions such as buy, exchange rate, finalize, mint token, seal, and VENSale.
- VEN Token Contract: This contract implements the ERC20 interface and additional functionality specific to the VEN token. It includes features like claiming bonuses, minting tokens, offering bonuses, sealing the contract, etc. Notable functions include mint, seal, offerBonus, claimBonus, transfer, and transferFrom.
- VENSale Contract: This contract manages the sale of VEN tokens. It defines various stages (Stage enum) such as Created, Initialized, Early, Normal, Closed, and Finalized. The sale has a start and end time (startTime and endTime) and includes logic for calculating the exchange rate (exchange-rate), determining the current stage (stage), buying tokens (buy), offering tokens to channels (offerToChannel), initializing the sale (initialize), and finalizing the sale (finalize).
- Overall, this contract facilitates the sale of VEN tokens in different stages, manages ownership and token functionality, and ensures safe mathematical operations. It adheres to the ERC20 standard for token functionality.

Audit scope

Name	Code Review and Security Analysis Report for VeChain Token Smart Contract
Platform	**Ethereum**
File	VEN.sol
Smart Contract Code	0xd850942ef8811f2a866692a623011bde52a462c1
Audit Date	April 8th, 2024

Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
VEN Contract Tokenomics: • Name: VeChain Token • Symbol: VEN • Decimals: 18 • Total supply: 1 billion **Ownership control:** • Mint tokens and assign them to someone. • Offer a bonus to raw token holders. • Set owner to zero address, disable mint, and enable token transfer.	**YES, This is valid.** **We suggest renouncing ownership once the ownership functions are not needed. This is to make the smart contract 100% decentralized.**
VENSale Contract Tokenomics: • Total supply: 1 billion VEN • 9% for private ICO. • 23% for commercial plans. • 5% for the team. • 22% for operations. • Non-Public Supply: 59% • Public Supply: 41% • Each account can be bought once in 30 minutes. • Maximum Buy Eth Amount: 30 ether • The early stage Lasts 3 days **Ownership control:** • Manually offer tokens to the channel. • Initialize to prepare for sale. • Finalize stage.	**YES, This is valid.** **We suggest renouncing ownership once the ownership functions are not needed. This is to make the smart contract 100% decentralized.**

Audit Summary

According to the standard audit assessment, the Customer`s solidity-based smart contracts are **“Secured”**. Also, these contracts contain owner control, which does not make them fully decentralized.

We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.

All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit Overview section. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium, 1 low, and 10 very low level issues.

Investor Advice: A technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner-controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	The solidity version is not specified	Passed
	The solidity version is too old	Moderated
	Integer overflow/underflow	Passed
	Function input parameters lack check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Moderated
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage is not set	Moderated
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Business Risk Analysis

Category	Result
Buy Tax	0%
Sell Tax	0%
Cannot Buy	No
Cannot Sell	No
Max Tax	0%
Modify Tax	Not Detected
Fee Check	No
Is Honeypot	Not Detected
Trading Cooldown	Not Detected
Can Pause Trade?	No
Pause Transfer?	Not Detected
Max Tax?	No
Is it Anti-whale?	Not Detected
Is Anti-bot?	Not Detected
Is it a Blacklist?	Not Detected
Blacklist Check	No
Can Mint?	Yes
Is it a Proxy?	Not Detected
Can Take Ownership?	Yes
Hidden Owner?	Not Detected
Self Destruction?	Not Detected
Auditor Confidence	High

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Code Quality

This audit scope has 1 smart contract. Smart contracts contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in VeChain Token are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties/methods can be reused many times by other contracts in the VeChain Token.

The EtherAuthority team has no scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a VeChain Token smart contract code in the form of an Etherscan web link.

As mentioned above, code parts are well commented on. and the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries used in this smart contract infrastructure are based on well-known industry-standard open-source projects.

Apart from libraries, their functions are not used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Remove constructor	Refer Audit Findings
2	VENSale	write	Explicitly mark visibility in the function	Refer Audit Findings
3	exchangeRate	write	Explicitly mark visibility in the function	Refer Audit Findings
4	blockTime	write	Explicitly mark visibility in the function	Refer Audit Findings
5	stage	write	Explicitly mark visibility in the function	Refer Audit Findings
6	isContract	internal	Passed	No Issue
7	buy	write	Missing required error message, Explicitly mark visibility in function	Refer Audit Findings
8	officialSold	write	Explicitly mark visibility in the function	Refer Audit Findings
9	channelsSold	write	Explicitly mark visibility in the function	Refer Audit Findings
10	offerToChannel	write	Missing required error message, Explicitly mark visibility in function, Centralization	Refer Audit Findings
11	initialize	write	Missing required error message, Explicitly mark visibility in function, Centralization	Refer Audit Findings
12	finalize	write	Missing required error message, Explicitly mark visibility in function	Refer Audit Findings
13	receiveApproval	write	Explicitly mark visibility in the function	Refer Audit Findings
14	VEN	write	Explicitly mark visibility in function, Centralization, and Code Contains Empty Blocks	Refer Audit Findings
15	totalSupply	write	Explicitly mark visibility in the function	Refer Audit Findings
16	isSealed	write	Explicitly mark visibility in the function	Refer Audit Findings
17	lastMintedTimestamp	write	Explicitly mark visibility in the function	Refer Audit Findings
18	claimBonus	internal	Missing required error message	Refer Audit Findings
19	balanceOf	write	Explicitly mark visibility in the function	Refer Audit Findings

20	transfer	write	Missing required error message, Explicitly mark visibility in function	Refer Audit Findings
21	transferFrom	write	Missing required error message, Explicitly mark visibility in function	Refer Audit Findings
22	approve	write	Explicitly mark visibility in the function	Refer Audit Findings
23	approveAndCall	write	Explicitly mark visibility in the function	Refer Audit Findings
24	allowance	write	Explicitly mark visibility in the function	Refer Audit Findings
25	mint	write	Explicitly mark visibility in function, Unlimited token minting	Refer Audit Findings
26	offerBonus	write	Explicitly mark visibility in function, Centralization	Refer Audit Findings
27	seal	write	Explicitly mark visibility in function, Centralization	Refer Audit Findings
28	totalSupply	write	Explicitly mark visibility in the function	Refer Audit Findings
29	balanceOf	write	Explicitly mark visibility in the function	Refer Audit Findings
30	transfer	write	Explicitly mark visibility in the function	Refer Audit Findings
31	transferFrom	write	Explicitly mark visibility in the function	Refer Audit Findings
32	approve	write	Explicitly mark visibility in the function	Refer Audit Findings
33	allowance	write	Explicitly mark visibility in the function	Refer Audit Findings
34	onlyOwner	modifier	Missing required error message	Refer Audit Findings
35	Owned	write	Explicitly mark visibility in the function	Refer Audit Findings
36	setOwner	write	Missing zero address validation, Explicitly mark visibility in function, Centralization	Refer Audit Findings

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets, that can't have a significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium-severity vulnerabilities were found.

Low

(1) Unlimited token minting: **VEN.sol**

```
// Mint tokens and assign to someone
    function mint(address _owner, uint256 _amount, bool _isRaw,
uint32 timestamp) onlyOwner{
        if (_isRaw) {
            accounts[_owner].rawTokens =
_amount.add(accounts[_owner].rawTokens).toUINT112();
            supplies.rawTokens =
_amount.add(supplies.rawTokens).toUINT128();
        } else {
            accounts[_owner].balance =
_amount.add(accounts[_owner].balance).toUINT112();
        }

        accounts[_owner].lastMintedTimestamp = timestamp;

        supplies.total = _amount.add(supplies.total).toUINT128();
        Transfer(0, _owner, _amount);
    }
```

Token minting without any maximum limit is considered inappropriate for tokenomics.

Resolution: We recommend placing some limit on token minting to mitigate this issue.

Very Low / Informational / Best practices:

(1) Use the latest solidity version: **Owned.sol**

```
pragma solidity ^0.4.13;
```

Use the latest solidity version while contract deployment to prevent any compiler version-level bugs.

Resolution: Please use versions greater than 0.8.25.

(2) Missing required error message:

Owned.sol

```
modifier onlyOwner() {
        require(msg.sender == owner);
        _;
    }
```

VENSale.sol

- buy
- offerToChannel
- initialize
- finalize

VEN.sol

```
function transferFrom(
        address _from,
        address _to,
        uint256 _amount
    ) returns (bool success) {
        require(isSealed());

        // implicitly claim bonus for both sender and receiver
        claimBonus(_from);
```

```
        claimBonus(_to);

        // according to VEN's total supply, never overflow here
        if (accounts[_from].balance >= _amount
            && allowed[_from][msg.sender] >= _amount
            && _amount > 0) {
            accounts[_from].balance -= uint112(_amount);
            allowed[_from][msg.sender] -= _amount;
            accounts[_to].balance =
_amount.add(accounts[_to].balance).toUINT112();
            Transfer(_from, _to, _amount);
            return true;
        } else {
            return false;
        }
    }

      function transfer(address _to, uint256 _amount) returns (bool
success) {
        require(isSealed());

        // implicitly claim bonus for both sender and receiver
        claimBonus(msg.sender);
        claimBonus(_to);

        // according to VEN's total supply, never overflow here
        if (accounts[msg.sender].balance >= _amount
            && _amount > 0) {
            accounts[msg.sender].balance -= uint112(_amount);
            accounts[_to].balance =
_amount.add(accounts[_to].balance).toUINT112();
            Transfer(msg.sender, _to, _amount);
            return true;
        } else {
            return false;
        }
    }

// Claim bonus by raw tokens
```

```
    function claimBonus(address _owner) internal{
        require(isSealed());
        if (accounts[_owner].rawTokens != 0) {
            uint256 realBalance = balanceOf(_owner);
            uint256 bonus = realBalance
                .sub(accounts[_owner].balance)
                .sub(accounts[_owner].rawTokens);

            accounts[_owner].balance = realBalance.toUINT112();
            accounts[_owner].rawTokens = 0;
            if(bonus > 0){
                Transfer(this, _owner, bonus);
            }
        }
    }
```

There is no error message set in the required condition.

Resolution: We suggest setting relevant error messages to identify the failure of the transaction, Otherwise the user can't identify the actual problem of why the transaction is not successful.

(3) Missing zero address validation: **Owned.sol**

```
function setOwner(address _newOwner) onlyOwner {
        owner = _newOwner;
    }
```

Detects missing zero address validation.

Resolution: We suggest first Checking that the address is not zero.

(4) Explicitly mark visibility in function:

Token.sol

- totalSupply
- balanceOf
- transfer

- transferFrom
- approve
- allowance

Owned.sol

```
function setOwner(address _newOwner) onlyOwner {
        owner = _newOwner;
    }
function Owned() {
        owner = msg.sender;
    }
```

VENSale.sol

```
  /// @notice entry to buy tokens
    function () payable {
        buy();
    }

    /// @notice entry to buy tokens
    function buy() payable {
        // reject contract buyer to avoid breaking interval limit
        require(!isContract(msg.sender));
        require(msg.value >= 0.01 ether);
```

- VENSale
- exchangeRate
- blockTime
- stage
- officialSold
- channelsSold
- offerToChannel
- initialize
- finalize

VEN.sol

- VEN
- totalSupply
- isSealed
- lastMintedTimestamp
- balanceOf
- transfer
- transferFrom
- approve
- approveAndCall
- allowance
- mint
- offerBonus
- seal

ApprovalReceiver.sol

```
    function receiveApproval(address _from, uint256 _value, address
_tokenContract, bytes _extraData);
```

In the code under review, several functions lack explicit visibility modifiers, such as public, internal, external, or private, which can lead to unexpected behavior and may pose security risks.

Resolution: Suggest adding explicit visibility modifiers to all functions in the code, specifying the appropriate level of visibility based on their intended use.

(5) Centralization:

Some functions of this smart contract are only called by Onlyowner

VEN.sol

- mint
- offerBonus
- seal

VENSale.sol

- offerToChannel
- initialize

- finalize

Owned.sol

```
function setOwner(address _newOwner) onlyOwner {
        owner = _newOwner;
    }
```

- setOwner

Resolution: We suggest making your smart contract 100% decentralized.

(6) Code Contains Empty Blocks: **VEN.sol**

```
// Constructor
    function VEN() {
    }
```

During the code review process, it was identified that the code includes empty blocks, which are sections of code enclosed by curly braces {} that do not contain any statements or logic. These empty blocks serve no functional purpose and should be addressed.

Resolution: Suggest removing all empty blocks from the code to improve code quality and clarity.

(7) Missing Error Message for revert function: **VEN.sol**

```
 // Send back ether sent to me
    function () {
        revert();
    }
```

During the code review process, it was observed that the default revert() function is used without an accompanying error message. Reverting without an error message can make debugging and error handling challenging both for developers and users of the contract.

Resolution: Suggest adding informative error messages to revert() statements to provide a clear context for transaction failures.

(8) Explicit Visibility for State Variables Warning:

VEN.sol

```
Supplies supplies;
// Balances for each account
 mapping(address => Account) accounts;

 // Owner of account approves the transfer of an amount to another
account
 mapping(address => mapping(address => uint256)) allowed;

 // bonus that can be shared by raw tokens
 uint256 bonusOffered;
```

VENSale.sol

```
 bool initialized;
 bool finalized;
 SoldOut soldOut;

 uint256 constant privateSupply = totalSupply * 9 / 100;  // 9% for
private ICO
 uint256 constant commercialPlan = totalSupply * 23 / 100; // 23%
for commercial plan
 uint256 constant reservedForTeam = totalSupply * 5 / 100;  // 5%
for team
 uint256 constant reservedForOperations = totalSupply * 22 / 100; //
22 for operations
```

The warning is related to the visibility of state variables in your Solidity code.

Resolution: We recommend updating the code to explicitly mark the visibility of state variables using the internal or public keyword, depending on the intended visibility.

(9) Solidity constants naming conventions: **VEN.sol**

```
uint256 constant venPerEth = 3500;  // normal exchange rate
uint256 constant venPerEthEarlyStage = venPerEth + venPerEth * 15 / 100;  // early stage has 15% reward

uint constant minBuyInterval = 30 minutes; // each account can buy once in 30 minutes
uint constant maxBuyEthAmount = 30 ether;

VEN ven; // VEN token contract follows ERC20 standard

address ethVault; // the account to keep received ether
address venVault; // the account to keep non-public offered VEN tokens

uint public constant startTime = 1503057600; // time to start sale
uint public constant endTime = 1504180800;   // tiem to close sale
uint public constant earlyStageLasts = 3 days; // early bird stage lasts in seconds
```

```
using SafeMath for uint256;

uint256 public constant totalSupply         = (10 ** 9) * (10 ** 18); // 1 billion VEN, decimals set to 18

uint256 constant privateSupply              = totalSupply * 9 / 100;  // 9% for private ICO
uint256 constant commercialPlan             = totalSupply * 23 / 100; // 23% for commercial plan
uint256 constant reservedForTeam            = totalSupply * 5 / 100;  // 5% for team
uint256 constant reservedForOperations      = totalSupply * 22 / 100; // 22 for operations

// 59%
uint256 public constant nonPublicSupply     = privateSupply + commercialPlan + reservedForTeam + reservedForOp
// 41%
uint256 public constant publicSupply = totalSupply - nonPublicSupply;


uint256 public constant officialLimit = 64371825 * (10 ** 18);
```

```
/// VEN token, ERC20 compliant
contract VEN is Token, Owned {
    using SafeMath for uint256;

    string public constant name     = "VeChain Token";  //The Token's name
    uint8 public constant decimals = 18;               //Number of decimals of the smallest unit
    string public constant symbol   = "VEN";           //An identifier
```

Constants are defined in lowercase.

Resolution: Constants should be named with all capital letters with underscores separating words.

(10) Remove constructor: **VEN.sol**

```
// Constructor
    function VEN() {
    }
```

If no parameters are passed to the constructor then no need to define them, if you don't define one explicitly, Solidity provides a default constructor that simply does nothing.

Resolution: Suggest to remove empty constructor.

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet's private key would be compromised, then it would create trouble. The following are Admin functions:

VEN.sol

- mint: Mint tokens assigned to someone by the owner.
- offerBonus: Offer a bonus to raw tokens held by the owner.
- seal: Set owner to zero address, disable mint, and enable token transfer by the current owner.

VENSale.sol

- offerToChannel: Manually offer tokens to the channel by the owner.
- finalize: finalize by the owner.
- initialize: initialize to prepare for sale by the owner.

Owned. sol

- setOwner: The current owner can set the new owner's address.

To make the smart contract 100% decentralized, we suggest renouncing ownership of the smart contract once its function is completed.

Conclusion

We were given a contract code in the form of Etherscan web links. And we have used all possible tests based on given objects as files. We observed 1 low and 10 informational issues in the smart contracts. And those issues are not critical. So, **it's good to go for the production**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover the maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

The security state of the reviewed smart contract, based on standard audit procedure scope, is **"Secured".**

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of the systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and white box penetration testing. We look at the project's website to get a high-level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

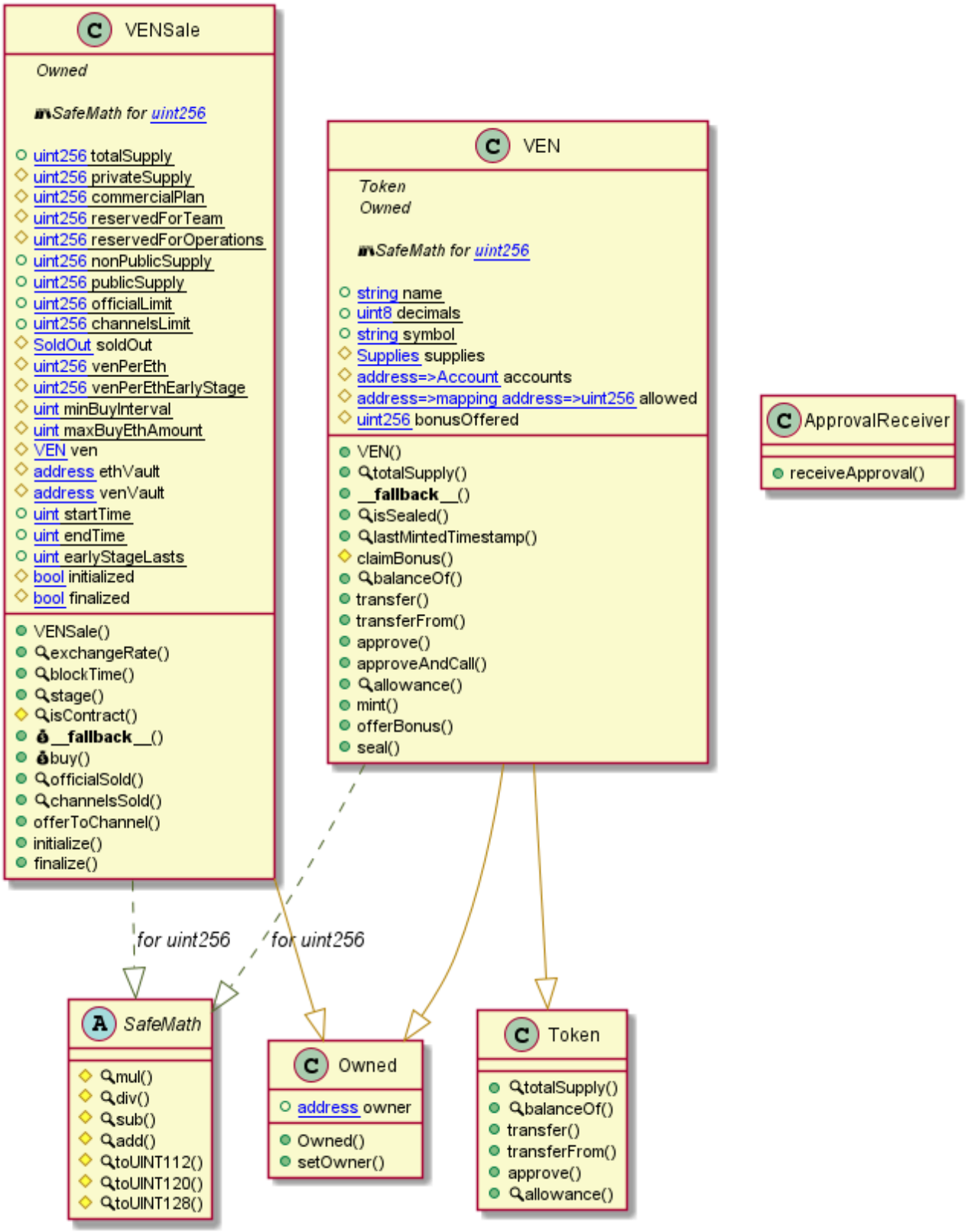
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - VeChain Token



Slither Results Log

Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We did the analysis of the project altogether. Below are the results.

Slither Log >> VEN.sol

```
Owned.setOwner(address) (VEN.sol#20-22) should emit an event for:
        - owner = _newOwner (VEN.sol#21)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control

Owned.setOwner(address)._newOwner (VEN.sol#20) lacks a zero-check on :
                - owner = _newOwner (VEN.sol#21)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in VENSale.initialize(VEN,address,address) (VEN.sol#544-577):
        External calls:
        - require(bool)(_ven.owner() == address(this)) (VEN.sol#551)
        State variables written after the call(s):
        - ethVault = _ethVault (VEN.sol#558)
        - ven = _ven (VEN.sol#556)
        - venVault = _venVault (VEN.sol#559)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in VENSale.buy() (VEN.sol#467-507):
        External calls:
        - require(bool)(blockTime() >= ven.lastMintedTimestamp(msg.sender) + minBuyInterval) (VEN.sol#476)
        - ven.mint(msg.sender,requested,true,blockTime()) (VEN.sol#494)
        External calls sending eth:
        - ethVault.transfer(ethCost) (VEN.sol#496)
        Event emitted after the call(s):
        - onSold(msg.sender,requested,ethCost) (VEN.sol#499)
Reentrancy in VENSale.finalize() (VEN.sol#580-594):
        External calls:
        - ven.offerBonus(unsold) (VEN.sol#588)
        - ven.seal() (VEN.sol#590)
        Event emitted after the call(s):
        - onFinalized() (VEN.sol#593)
Reentrancy in VENSale.initialize(VEN,address,address) (VEN.sol#544-577):
        External calls:
        - require(bool)(_ven.owner() == address(this)) (VEN.sol#551)
        - ven.mint(venVault,reservedForTeam.add(reservedForOperations),false,blockTime()) (VEN.sol#561-566)
        - ven.mint(venVault,privateSupply.add(commercialPlan),true,blockTime()) (VEN.sol#568-573)
        Event emitted after the call(s):
        - onInitialized() (VEN.sol#576)
Reentrancy in VENSale.offerToChannel(address,uint256) (VEN.sol#520-538):
        External calls:
        - ven.mint(_channelAccount,_venAmount,true,blockTime()) (VEN.sol#530-535)
        Event emitted after the call(s):
        - onSold(_channelAccount,_venAmount,0) (VEN.sol#537)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

VENSale.stage() (VEN.sol#418-450) uses timestamp for comparisons
        Dangerous comparisons:
        - blockTime() < startTime (VEN.sol#428)
        - blockTime() < endTime (VEN.sol#438)
        - blockTime() < startTime.add(earlyStageLasts) (VEN.sol#440)
VENSale.buy() (VEN.sol#467-507) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool)(blockTime() >= ven.lastMintedTimestamp(msg.sender) + minBuyInterval) (VEN.sol#476)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

VENSale.isContract(address) (VEN.sol#452-459) uses assembly
        - INLINE ASM (VEN.sol#455-458)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Pragma version^0.4.11 (VEN.sol#5) allows old versions
solc-0.4.13 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Parameter Owned.setOwner(address)._newOwner (VEN.sol#20) is not in mixedCase
Parameter VEN.lastMintedTimestamp(address)._owner (VEN.sol#179) is not in mixedCase
Parameter VEN.claimBonus(address)._owner (VEN.sol#184) is not in mixedCase
Parameter VEN.balanceOf(address)._owner (VEN.sol#201) is not in mixedCase
Parameter VEN.transfer(address,uint256)._to (VEN.sol#219) is not in mixedCase
Parameter VEN.transfer(address,uint256)._amount (VEN.sol#219) is not in mixedCase
Parameter VEN.transferFrom(address,address,uint256)._from (VEN.sol#245) is not in mixedCase
Parameter VEN.transferFrom(address,address,uint256)._to (VEN.sol#246) is not in mixedCase
Parameter VEN.transferFrom(address,address,uint256)._amount (VEN.sol#247) is not in mixedCase
Parameter VEN.approve(address,uint256)._spender (VEN.sol#271) is not in mixedCase
```

```
Parameter VEN.approveAndCall(address,uint256,bytes)._spender (VEN.sol#278) is not in mixedCase
Parameter VEN.approveAndCall(address,uint256,bytes)._value (VEN.sol#278) is not in mixedCase
Parameter VEN.approveAndCall(address,uint256,bytes)._extraData (VEN.sol#278) is not in mixedCase
Parameter VEN.allowance(address,address)._owner (VEN.sol#290) is not in mixedCase
Parameter VEN.allowance(address,address)._spender (VEN.sol#290) is not in mixedCase
Parameter VEN.mint(address,uint256,bool,uint32)._owner (VEN.sol#295) is not in mixedCase
Parameter VEN.mint(address,uint256,bool,uint32)._amount (VEN.sol#295) is not in mixedCase
Parameter VEN.mint(address,uint256,bool,uint32)._isRaw (VEN.sol#295) is not in mixedCase
Parameter VEN.offerBonus(uint256)._bonus (VEN.sol#310) is not in mixedCase
Event VENSaleonInitialized() (VEN.sol#596) is not in CapWords
Event VENSaleonFinalized() (VEN.sol#597) is not in CapWords
Event VENSaleonSold(address,uint256,uint256) (VEN.sol#599) is not in CapWords
Parameter VENSale.isContract(address)._addr (VEN.sol#452) is not in mixedCase
Parameter VENSale.offerToChannel(address,uint256)._channelAccount (VEN.sol#520) is not in mixedCase
Parameter VENSale.offerToChannel(address,uint256)._venAmount (VEN.sol#520) is not in mixedCase
Parameter VENSale.initialize(VEN,address,address)._ven (VEN.sol#545) is not in mixedCase
Parameter VENSale.initialize(VEN,address,address)._ethVault (VEN.sol#546) is not in mixedCase
Parameter VENSale.initialize(VEN,address,address)._venVault (VEN.sol#547) is not in mixedCase
Constant VENSale.totalSupply (VEN.sol#348) is not in UPPER_CASE_WITH_UNDERSCORES
Constant VENSale.privateSupply (VEN.sol#350) is not in UPPER_CASE_WITH_UNDERSCORES
Constant VENSale.commercialPlan (VEN.sol#351) is not in UPPER_CASE_WITH_UNDERSCORES
Constant VENSale.reservedForTeam (VEN.sol#352) is not in UPPER_CASE_WITH_UNDERSCORES
Constant VENSale.reservedForOperations (VEN.sol#353) is not in UPPER_CASE_WITH_UNDERSCORES
Constant VENSale.nonPublicSupply (VEN.sol#356) is not in UPPER_CASE_WITH_UNDERSCORES
Constant VENSale.publicSupply (VEN.sol#358) is not in UPPER_CASE_WITH_UNDERSCORES
Constant VENSale.officialLimit (VEN.sol#361) is not in UPPER_CASE_WITH_UNDERSCORES
Constant VENSale.channelsLimit (VEN.sol#362) is not in UPPER_CASE_WITH_UNDERSCORES
Constant VENSale.venPerEth (VEN.sol#377) is not in UPPER_CASE_WITH_UNDERSCORES
Constant VENSale.venPerEthEarlyStage (VEN.sol#378) is not in UPPER_CASE_WITH_UNDERSCORES
Constant VENSale.minBuyInterval (VEN.sol#380) is not in UPPER_CASE_WITH_UNDERSCORES
Constant VENSale.maxBuyEthAmount (VEN.sol#381) is not in UPPER_CASE_WITH_UNDERSCORES
Constant VENSale.startTime (VEN.sol#388) is not in UPPER_CASE_WITH_UNDERSCORES
Constant VENSale.endTime (VEN.sol#389) is not in UPPER_CASE_WITH_UNDERSCORES
Constant VENSale.earlyStageLasts (VEN.sol#390) is not in UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

```
Reentrancy in VENSale.buy() (VEN.sol#467-507):
        External calls:
        - ethVault.transfer(ethCost) (VEN.sol#496)
        State variables written after the call(s):
        - soldOut.official = requested.add(soldOut.official).toUINT120() (VEN.sol#498)
        Event emitted after the call(s):
        - onSold(msg.sender,requested,ethCost) (VEN.sol#499)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4

approveAndCall(address,uint256,bytes) should be declared external:
        - VEN.approveAndCall(address,uint256,bytes) (VEN.sol#278-288)
receiveApproval(address,uint256,address,bytes) should be declared external:
        - ApprovalReceiver.receiveApproval(address,uint256,address,bytes) (VEN.sol#323)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
VEN.sol analyzed (6 contracts with 84 detectors), 63 result(s) found
```

Solidity Static Analysis

Static code analysis is used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it.

VEN.sol

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in VENSale.buy(): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 467:4:

Inline assembly:

The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.

more

Pos: 455:8:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

more

Pos: 413:22:

Gas costs:

Fallback function of contract VENSale requires too much gas (infinite). If the fallback function requires more than 2300 gas, the contract cannot receive Ether.

Pos: 170:4:

Gas costs:

Gas requirement of function VEN.mint is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 295:4:

Constant/View/Pure functions:

VEN.lastMintedTimestamp(address) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 179:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 582:8:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 378:55:

Solhint Linter

Linters are the utility tools that analyze the given source code and report programming errors, bugs, and stylistic errors. For the Solidity language, there are some linter tools available that a developer can use to improve the quality of their Solidity contracts.

VEN.sol

```
Compiler version ^0.4.13 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:4
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:10
Provide an error message for require
Pos: 9:15
Explicitly mark visibility in function
Pos: 5:19
Explicitly mark visibility in function
Pos: 5:85
Explicitly mark visibility in function
Pos: 5:89
Explicitly mark visibility in function
Pos: 5:95
Explicitly mark visibility in function
Pos: 5:102
Explicitly mark visibility in function
Pos: 5:108
Explicitly mark visibility in function
Pos: 5:113
Constant name must be in capitalized SNAKE_CASE
Pos: 5:124
Constant name must be in capitalized SNAKE_CASE
Pos: 5:125
Constant name must be in capitalized SNAKE_CASE
Pos: 5:126
Explicitly mark visibility of state
Pos: 5:136
Explicitly mark visibility of state
Pos: 5:152
Explicitly mark visibility of state
Pos: 5:155
Explicitly mark visibility of state
Pos: 5:158
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:161
Code contains empty blocks
Pos: 20:161
Explicitly mark visibility in function
Pos: 5:164
When fallback is not payable you will not be able to receive ether
```

```
Pos: 5:169
Explicitly mark visibility in function
Pos: 5:169
Provide an error message for revert
Pos: 9:170
Explicitly mark visibility in function
Pos: 5:174
Explicitly mark visibility in function
Pos: 5:178
Provide an error message for require
Pos: 9:184
Explicitly mark visibility in function
Pos: 5:200
Explicitly mark visibility in function
Pos: 5:218
Provide an error message for require
Pos: 9:219
Explicitly mark visibility in function
Pos: 5:243
Provide an error message for require
Pos: 9:248
Explicitly mark visibility in function
Pos: 5:270
Explicitly mark visibility in function
Pos: 5:277
Explicitly mark visibility in function
Pos: 5:289
Explicitly mark visibility in function
Pos: 5:294
Explicitly mark visibility in function
Pos: 5:309
Explicitly mark visibility in function
Pos: 5:316
Explicitly mark visibility in function
Pos: 5:322
Constant name must be in capitalized SNAKE_CASE
Pos: 5:347
Explicitly mark visibility of state
Pos: 5:349
Constant name must be in capitalized SNAKE_CASE
Pos: 5:349
Explicitly mark visibility of state
Pos: 5:350
Constant name must be in capitalized SNAKE_CASE
Pos: 5:350
Explicitly mark visibility of state
Pos: 5:351
Constant name must be in capitalized SNAKE_CASE
Pos: 5:351
Explicitly mark visibility of state
Pos: 5:352
Constant name must be in capitalized SNAKE_CASE
Pos: 5:352
Constant name must be in capitalized SNAKE_CASE
Pos: 5:355
Constant name must be in capitalized SNAKE_CASE
Pos: 5:357
Constant name must be in capitalized SNAKE_CASE
Pos: 5:360
```

```
Constant name must be in capitalized SNAKE_CASE
Pos: 5:361
Explicitly mark visibility of state
Pos: 5:374
Explicitly mark visibility of state
Pos: 5:376
Constant name must be in capitalized SNAKE_CASE
Pos: 5:376
Explicitly mark visibility of state
Pos: 5:377
Constant name must be in capitalized SNAKE_CASE
Pos: 5:377
Explicitly mark visibility of state
Pos: 5:379
Constant name must be in capitalized SNAKE_CASE
Pos: 5:379
Explicitly mark visibility of state
Pos: 5:380
Constant name must be in capitalized SNAKE_CASE
Pos: 5:380
Explicitly mark visibility of state
Pos: 5:382
Explicitly mark visibility of state
Pos: 5:384
Explicitly mark visibility of state
Pos: 5:385
Constant name must be in capitalized SNAKE_CASE
Pos: 5:387
Constant name must be in capitalized SNAKE_CASE
Pos: 5:388
Constant name must be in capitalized SNAKE_CASE
Pos: 5:389
Explicitly mark visibility of state
Pos: 5:391
Explicitly mark visibility of state
Pos: 5:392
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:394
Explicitly mark visibility in function
Pos: 5:400
Explicitly mark visibility in function
Pos: 5:411
Avoid making time-based decisions in your business logic
Pos: 23:412
Explicitly mark visibility in function
Pos: 5:417
Visibility modifier must be first in list of modifiers
Pos: 49:451
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:454
Explicitly mark visibility in function
Pos: 5:461
Explicitly mark visibility in function
Pos: 5:466
Provide an error message for require
Pos: 9:468
Provide an error message for require
Pos: 9:469
```

```
Provide an error message for require
Pos: 9:473
Provide an error message for require
Pos: 9:475
Possible reentrancy vulnerabilities. Avoid state changes after
transfer.
Pos: 13:497
Explicitly mark visibility in function
Pos: 5:509
Explicitly mark visibility in function
Pos: 5:514
Explicitly mark visibility in function
Pos: 5:519
Provide an error message for require
Pos: 9:522
Provide an error message for require
Pos: 9:527
Explicitly mark visibility in function
Pos: 5:543
Provide an error message for require
Pos: 9:547
Provide an error message for require
Pos: 9:550
Provide an error message for require
Pos: 9:552
Provide an error message for require
Pos: 9:553
Explicitly mark visibility in function
Pos: 5:579
Provide an error message for require
Pos: 9:581
Event name must be in CamelCase
Pos: 5:595
Event name must be in CamelCase
Pos: 5:596
Event name must be in CamelCase
Pos: 5:598
```

Software analysis result:

These software reported many false positive results and some are informational issues. So, those issues can be safely ignored.

Email: audit@EtherAuthority.io