verichains

SECURITY AUDIT OF

TRUSTK AND TOKENSALE SMART CONTRACT

Public Report

Oct 21, 2021

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Oct 15, 2021. We would like to thank the TrustKeys for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the TRUSTK and TokenSale Smart Contract. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified no vulnerable issues in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About TrustKeys

TrustKeys Network is a Blockchain ecosystem that includes: Decentralized identity social network, cryptocurrency exchange and digital asset storage wallet. TrustKeys Network was created to help the community have the most complete Blockchain application that meets the needs of investing, storing assets, exchanging information safely and reliably.

1.2. About TRUSTK and TokenSale Smart Contract

TRUSTK is a smart contract which implements ERC20 token.

TokenSale is a smart contract which allows buying TrustKeys tokens by a specific token.

1.3. Audit scope

This audit focused on identifying security flaws in code and the design of two smart contracts:

- TRUSTK smart contract at the repo link:

https://github.com/trustkeys/trustk/blob/master/bep20-erc20/

It was conducted on commit fdf812ec2a5959fa45721cb80a5940f5f8af00e3.

- TokenSale's TRUSTK at the repo link:

https://github.com/trustkeys/trustk/tree/master/IDO/SaleFixedPrice

It was conducted on commit 6b346861629ccb6e69841990bd69119b526bd8a6.

1.4. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert

- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.5. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The initial review was conducted in Oct 2021 and a total effort of 3 working days was dedicated to identifying and documenting security issues in the code base of TRUSTK and TokenSale Smart Contract.

The following files were made available in the course of the review:

SHA256 SUM	FILE
59933b0c7c7e1b2726b438aa9f359f789908f46654d4fe0a2c4261c60e3958d2	TRUSTK.sol
89cf3ea0215029b05f335b893845bfcfe3d32860a3ade8808fdd4ba02114e81c	Context.sol
90b144c66bab8a58df119316c959c9d06aef952de2dbc047a96db81241d3208e	IERC20.sol
68afbef712cb3b326500e3ee338143989bd9001aebbc9e6016f420440fd3df3b	IERC20Metadata.sol
8cfae6e8e82630c885c1297e509ed6f50e130e462364b1c004b4a21dee7c3ab2	Ownable.sol
276e0f1ea163c137e25597e3c8929b9076fc1df355c73f1ebfea3133d285302e	Sale.sol

2.2. Findings

The audit team found no vulnerability in the given version of TRUSTK and TokenSale Smart Contract.

2.3. Additional information and recommendations

2.3.1. Unnecessary condition checks INFORMATIVE

There are a lot of if-statements in the file that are unnecessary because they check variables covered by SafeMath method.

In Sale.sol, there are 2 require statements that are not necessary:

```
88  require(tokenAccept.allowance(msg.sender,address(this)) >= amount…
    );
```

```
 89
 90  require(tokenAccept.transferFrom(msg.sender, owner(), amount));
 ...
117  function updateMinBuy(uint256 _minBuy) public onlyOwner() {
118      require(_minBuy >= 0);
```

Snippet 1. Sale.sol unnecessary requires

The require statement on line 88 will be checked by the token call on line 90. The require statement on line 118 can never be triggered.

```
 46  constructor(IERC20Metadata _tokenSale, uint256 _numTokenSale, IERC20…
     Metadata _tokenAccept, uint256 _numTokenAccept, uint256 _minBuy…
     ,  bool _isActive) {
 47      require(_numTokenSale > 0);
 48      require(_numTokenAccept > 0);
 49      require(_minBuy >= 0);
 ...
 77  function transfer(address _to, uint256 _value) {
 78      if (_to == 0x0) throw;                                // Prevent …
     transfer to 0x0 address. Use burn() instead
 79      if (_value <= 0) throw;
 80      if (balanceOf[msg.sender] < _value) throw;           // Check…
      if the sender has enough
 81      if (balanceOf[_to] + _value < balanceOf[_to]) throw; // Check…
      for overflows
 82      balanceOf[msg.sender] = SafeMath.safeSub(balanceOf[msg.sender], …
     _value);                     // Subtract from the sender
 83      balanceOf[_to] = SafeMath.safeAdd(balanceOf[_to], _value);      …
                              // Add the same to the recipient
 84      Transfer(msg.sender, _to, _value);                   // Notify a…
     nyone listening that this transfer took place
 85  }
 ...
 97  function transferFrom(address _from, address _to, uint256 _value) re…
     turns (bool success) {
 98      if (_to == 0x0) throw;                                // Prevent…
      transfer to 0x0 address. Use burn() instead
 99      if (_value <= 0) throw;
100      if (balanceOf[_from] < _value) throw;                // Chec…
     k if the sender has enough
101      if (balanceOf[_to] + _value < balanceOf[_to]) throw;  // Chec…
     k for overflows
102      if (_value > allowance[_from][msg.sender]) throw;     // Chec…
     k allowance
```

```
103     balanceOf[_from] = SafeMath.safeSub(balanceOf[_from], _value);  …
                                // Subtract from the sender
104     balanceOf[_to] = SafeMath.safeAdd(balanceOf[_to], _value);      …
                              // Add the same to the recipient
105     allowance[_from][msg.sender] = SafeMath.safeSub(allowance[_from]…
    [msg.sender], _value);
___  ____________________________________________________________________
110 function burn(uint256 _value) returns (bool success) {
111     if (balanceOf[msg.sender] < _value) throw;              // Chec…
    k if the sender has enough
112     if (_value <= 0) throw;
113     balanceOf[msg.sender] = SafeMath.safeSub(balanceOf[msg.sender], …
    _value);                          // Subtract from the sender
___  ____________________________________________________________________
119 function freeze(uint256 _value) returns (bool success) {
120     if (balanceOf[msg.sender] < _value) throw;              // Chec…
    k if the sender has enough
121     if (_value <= 0) throw;
122     balanceOf[msg.sender] = SafeMath.safeSub(balanceOf[msg.sender], …
    _value);                          // Subtract from the sender
___  ____________________________________________________________________
128 function unfreeze(uint256 _value) returns (bool success) {
129     if (freezeOf[msg.sender] < _value) throw;               // Check…
     if the sender has enough
130     if (_value <= 0) throw;
131     freezeOf[msg.sender] = SafeMath.safeSub(freezeOf[msg.sender], _v…
    alue);                          // Subtract from the sender
```

Snippet 2. TRUSTK.sol unnecessary condition checks

In TRUSTK.sol, many if statements's condition will be checked again by their following SafeMath operators. Specially, they're on line 80, 81, 100, 101, 102, 111, 120 and 129. The require statement on line 49 can never be triggered.

RECOMMENDATION

We suggest removing those condition checks for gas saving.

2.3.2. Outdated version of Solidity INFORMATIVE

The required version of Solidity in the TRUSTK contract source code is 0.4.8, which is outdated.

Updating the Solidity version from 0.4.8 to 0.8.0 will introduce many features like:

- Automatic integer overflow/underflow checking for arithmetic operations.

- Abstract contract support.
- New patterns for error handling.

2.3.3. Typo in getRemaningTokenSale function INFORMATIVE

There is a typo in function name in file Sale.sol, the correct function name should be getRemainingTokenSale

```
123  function getRemaningTokenSale() public view returns (uint256) {
```

Snippet 3. Sale.sol typo

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Oct 15, 2021*	Private Report	Verichains Lab
1.1	*Oct 21, 2021*	Public Report	Verichains Lab

Table 2. Report versions history