verichains

SECURITY AUDIT OF

ISTEP TOKEN SMART CONTRACT

Public Report

May 12, 2022

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 May 12, 2022. We would like to thank the iStep 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 iStep Token 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 some vulnerable issues in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About iStep Token Smart Contract

iSTEP is a lifestyle app featuring aspects of Social-Fi and Game-Fi.

Users using NFT Sneakers can earn ISTEP by walking, jogging, or running outdoors. ISTEP can be used to level up and buy new Sneakers.We aim to build a huge running community and promote the spirit of improving physical health together by awarding tokens to runners and creating tournaments.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the iStep Token Smart Contract.

It was conducted on commit 9b86dead4e525b203c39d3a24b011d98b32e69c0 from git repository *https://github.com/istepofficial/token-smartcontract*.

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

SHA256 Sum	File
8a342cb867cc84caf16bd6d74d24ca237d617c73dd59289462a36e97b25a9c96	**TokenISTEP.sol**

1.3. 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

1.4. 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 iStep Token Smart Contract was written in Solidity language, with the required version to be ^0.8.0. The source code was written based on OpenZeppelin's library.

2.1.1. TokenISTEP contract

TokenISTEP is an upgradable ERC20 token contract which extends OwnableUpgradeable and ERC20Upgradeable contracts. With OwnableUpgradeable, by default, Token Owner is contract deployer, but he can transfer ownership to another address at any time. Please note that with ERC20Upgradeable, the whole contract can be upgraded anytime to change any logic (mint, transfer, lock, ...).

The contract pre-mints 300,000,000 tokens for the owner when initializing. The Token Owner can set any addresses to be operator. Operators can mint any amount of tokens, burn any amount of tokens from any addresses and update unlockPercent. The operators can also transfer locked amount of tokens to users. Users can only transfer unlockPercent of their lockBalances. If users approve for operators to use their tokens, operators can transfer and unlock tokens for users.

Table below lists some properties of the audited iStep Token Smart Contract (as of the report writing time).

PROPERTY	VALUE
Name	ISTEP
Symbol	ISTEP
Decimals	18
Max Supply	Unlimited

Table 2. The iStep Token Smart Contract properties

2.2. Findings

During the audit process, the audit team had identified some vulnerable issues in the smart contracts code.

iStep fixed the code, according to Verichains's private report, in commit c106d7dd626f4d31a509bfc365fba0d96a0f8a44.

2.2.1. Invalid unlocked balance check in transferFrom HIGH

In transferFrom function, it is required that balanceOf(from) - locked > amount so the users can not transfer all their unlocked balance.

```
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
    if (unlockPercent < 100) {
        uint256 locked = (lockBalances[from] * (100 - unlockPercent)) / 1…
  00;
        require(balanceOf(from) - locked  >  amount, "Unlocked balance is…
   not enough");
    }

    address spender = _msgSender();
    _spendAllowance(from, spender, amount);
    _transfer(from, to, amount);
    return true;
}
```

RECOMMENDATION

Changing > to >=.

```
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
    if (unlockPercent < 100) {
        uint256 locked = (lockBalances[from] * (100 - unlockPercent)) / 1…
  00;
        require(balanceOf(from) - locked  >=  amount, "Unlocked balance i…
  s not enough");
    }

    address spender = _msgSender();
    _spendAllowance(from, spender, amount);
    _transfer(from, to, amount);
```

```
    return true;
}
```

UPDATES

- *May 12, 2022*: This issue has been acknowledged and fixed by the iStep team.

2.2.2. Wrong unlock calculation in transferFromAndUnlock INFORMATIVE

In transferFromAndUnlock function, transfer amount is compared to locked so the unlocked amount should be calculation from locked instead of lockBalances.

For example with current calculation:

lockBalances[from] is 1000 and unlockPercent is 50, locked will be 500.

If transferFromAndUnlock 500 tokens, lockBalances[from] will be 0, locked balance will be 0.

If transferFromAndUnlock 499 tokens, lockBalances[from] will be 501, locked balance will be 250.5 which should be 1.

```
function transferFromAndUnlock(
    address from,
    address to,
    uint256 amount
) public onlyOperatorAccount returns (bool) {
    address spender = _msgSender();
    _spendAllowance(from, spender, amount);
    _transfer(from, to, amount);

    uint256 locked = (lockBalances[from] * (100 - unlockPercent)) / 100;
    if (locked > amount) {
        lockBalances[from] = lockBalances[from] - amount;
    } else {
        lockBalances[from] = 0;
    }
    return true;
}
```

RECOMMENDATION

Changing the code like below.

```
function transferFromAndUnlock(
    address from,
```

```
    address to,
    uint256 amount
) public onlyOperatorAccount returns (bool) {
    address spender = _msgSender();
    _spendAllowance(from, spender, amount);
    _transfer(from, to, amount);

    uint256 locked = (lockBalances[from] * (100 - unlockPercent)) / 100;
    if (locked > amount) {
        lockBalances[from] = (locked - amount) * 100 / unlockPercent;
    } else {
        lockBalances[from] = 0;
    }
    return true;
}
```

UPDATES

- *May 12, 2022*: This issue has been acknowledged and fixed by the iStep team.

2.2.3. Remove unused import "hardhat/console.sol"; INFORMATIVE

Consider remove unused import "hardhat/console.sol";

UPDATES

- *May 12, 2022*: This issue has been acknowledged and fixed by the iStep team.

APPENDIX

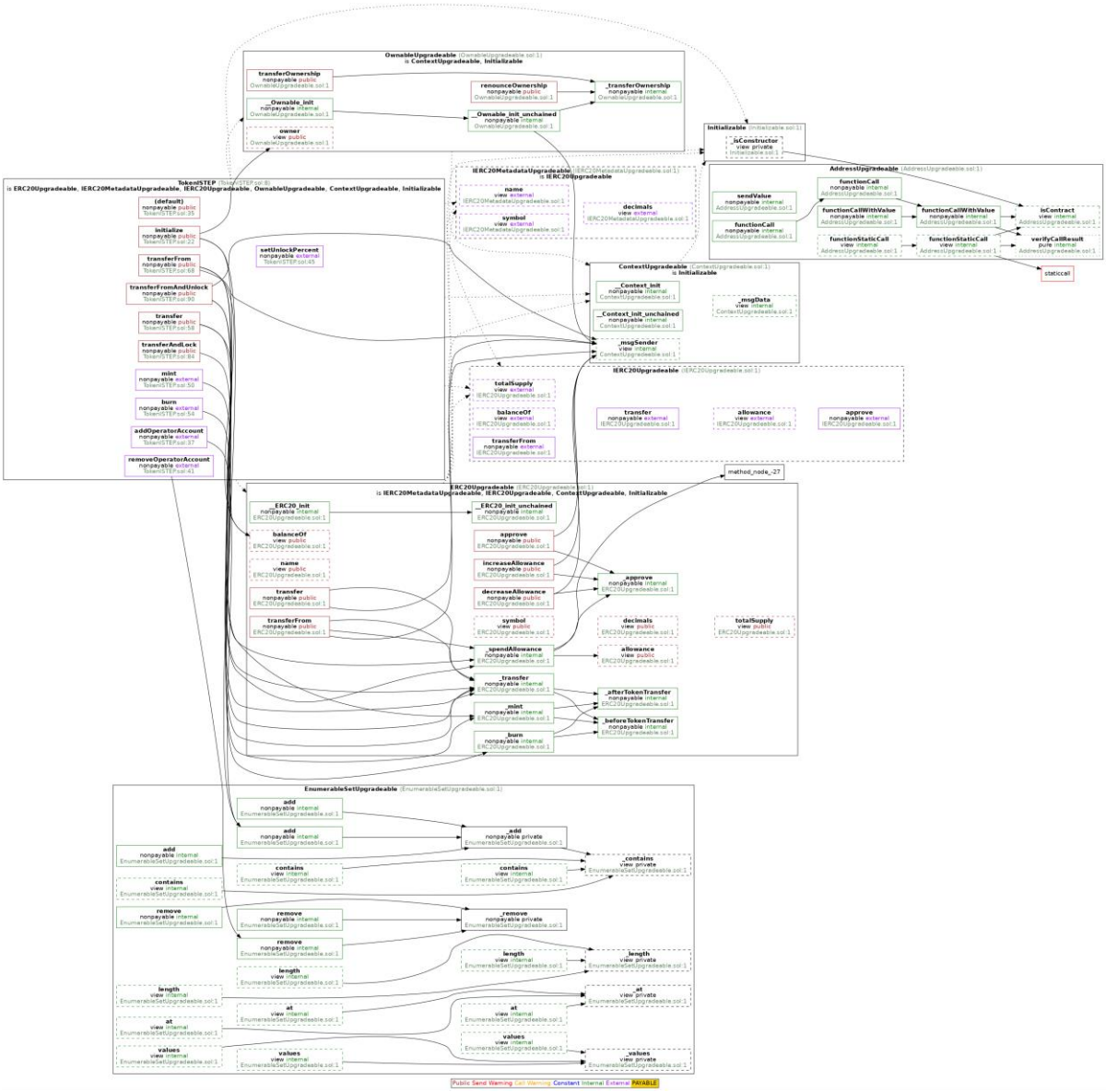

Image 1. iStep Token Smart Contract call graph

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*May 10, 2022*	Private Report	Verichains Lab
1.1	*May 12, 2022*	Public Report	Verichains Lab

Table 3. Report versions history