SECURITY AUDIT OF

BUSINESS AGE OF EMPIRES SMART CONTRACTS

Public Report

Jan 24, 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 Jan 24, 2022. We would like to thank the BAoE 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 Business Age of Empires Smart Contracts. 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 minor vulnerable issues in the contract code. BAoE team has resolved and updated all the recommendations.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Business Age of Empires Smart Contracts

BAoE is a "PLAY TO EARN" game built on the BSC platform. In Business Age of Empires, players will embody cyborgs, go on an adventure to uncover the treasures, and learn more about mankind's once-famous civilizations.

Business Age of Empires features basic yet appealing gameplay that is appropriate for all ages, and the game also demands players to think creatively in order to find a variety of priceless gems. These are the significant benefits that contribute to the game's unique appeal. Being enveloped in the rapid growth of Blockchain technology as well as the NFT coding trend, which is quickly entering the conventional gaming industry with a big number of Crypto believers.

Business Age of Empires promises to make significant advancements in the project's development as well as the day-by-day completion of the full Ecosystem established by our developers.

Binance Smart Chain technology (BSC/BEP20) is used to build the game's platform. In-game objects are encrypted in NFT format to allow for P2P peer-to-peer transactions and to support the Play to Earn (P2E) system.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the smart contracts of Business Age of Empires Smart Contracts. It was conducted on the source code provided by the team.

It was conducted on commit ee1d14cb11da45b404b665210d862373410e4307 from git repository *https://github.com/b-aoe/mining-simulator/*.

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 initial review was conducted on Jan 21, 2022 and a total effort of 3 working days was dedicated to identifying and documenting security issues in the code base of the Business Age of Empires Smart Contracts.

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

SHA256 Sum	File
963a92b15c00a8013af530d9cd17f94936b2685b68733630e6691b5378e630b9	**IDO.sol**
4746fca817080d1ccf28e61bf3fdb62a491b510c29cfb169e794e63ed73c14e6	**Uniswap.sol**
125c51517b29e1e760f633356fadb75640040e5cf27dc1c279dc788a0890dab3	**Token.sol**

2.2. Contract code

The Business Age of Empires Smart Contracts was written in Solidity language, with the required version to be ^0.8.0. The source code was written based on OpenZeppelin's library.

The provided source codes consist of two main parts which use some libraries and contracts from OpenZeppelin.

2.2.1. Token contract

Below are some basic properties of the BAoE token contract.

PROPERTY	VALUE
Name	BAoE
Symbol	BAoE
Decimals	18
Total Supply	100,000,000 (x10^{18}) Note: the number of decimals is 18, so the total representation token will be 100,000,000 or 100 million.

Table 2. The Business Age of Empires Smart Contracts properties

2.2.2. IDO contract

The IDO contract is used for the token distribution process, this process can be summarized as below:

- Before the start time, all tokens are locked in the IDO contract without TGE.
- Once the cliff time is reached, all the tokens will be unlocked at the end of each period.

2.3. Findings

During the audit process, the audit team found no vulnerability in the given version of Business Age of Empires Smart Contracts.

2.4. Additional notes and recommendations

2.4.1. IDO.sol - firstReturn and periodReturn should be the same precision INFORMATIVE

The firstReturn and periodReturn variables are used to represent the percentages of the claim amounts. These two variables should have the same precision (100 or 1000) to avoid some errors when inputting these values through the constructor.

```
function getClaimableAmount(address receiver) public returns (uint256) {
    if (block.timestamp < startTime+cliff) {
        return 0;
    }
    uint256 currentPeriod = (block.timestamp-(startTime+cliff))/timePerPe…
  riod;

    if (currentPeriod > totalPeriods) {
        currentPeriod = totalPeriods;
    }
    uint256 claimAmount = userFunds[receiver] * firstReturn/100;
    claimAmount += currentPeriod * userFunds[receiver] * periodReturn/100…
  00;
    if (claimAmount > userFunds[receiver]) {
        claimAmount = userFunds[receiver];
    }
    return claimAmount - userClaimed[receiver];
}
```

RECOMMENDATION

Change the precision of the firstReturn variable to 1000 for consistency with the periodReturn variable.

2.4.2. IDO.sol - Missing length check between accounts and packages array INFORMATIVE

The accounts and packages array should be checked to ensure that they have the same length before processing.

```
constructor(
    address _token,
    uint256 _startTime,
    uint256 _cliff,
    uint256 _totalPeriods,
    uint256 _timePerPeriod,
    uint256 _firstReturn,
    uint256 _periodReturn,
    address[] memory accounts,
    uint256[] memory packages
) {
    BATK = BAoE(_token);
    firstReturn = _firstReturn;
    periodReturn = _periodReturn;
    startTime = _startTime;
    cliff = _cliff;
    totalPeriods = _totalPeriods;
    timePerPeriod = _timePerPeriod;
    adminlist[msg.sender] = 1;
    for (uint256 i = 0; i < accounts.length; i++) {
        userFundsInUSDT[accounts[i]] += packages[i]*(10**18);
        userFunds[accounts[i]] += (packages[i]*(10**18) * 40);
        userClaimed[accounts[i]] = 0;
        userRemain[accounts[i]] = userFunds[accounts[i]];
        userToken[numberOfAccounts] = accounts[i];
        numberOfAccounts = numberOfAccounts + 1;
    }

}
```

RECOMMENDATION

The code can be fixed as below.

```
constructor(
    address _token,
    uint256 _startTime,
    uint256 _cliff,
    uint256 _totalPeriods,
    uint256 _timePerPeriod,
    uint256 _firstReturn,
    uint256 _periodReturn,
    address[] memory accounts,
    uint256[] memory packages
) {
    require(accounts.length == packages.length, "INVALID PARAMETER");
    BATK = BAoE(_token);
    firstReturn = _firstReturn;
    periodReturn = _periodReturn;
    startTime = _startTime;
    cliff = _cliff;
    totalPeriods = _totalPeriods;
    timePerPeriod = _timePerPeriod;
    adminlist[msg.sender] = 1;
    for (uint256 i = 0; i < accounts.length; i++) {
        userFundsInUSDT[accounts[i]] += packages[i]*(10**18);
        userFunds[accounts[i]] += (packages[i]*(10**18) * 40);
        userClaimed[accounts[i]] = 0;
        userRemain[accounts[i]] = userFunds[accounts[i]];
        userToken[numberOfAccounts] = accounts[i];
        numberOfAccounts = numberOfAccounts + 1;
    }

}
```

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Jan 24, 2022*	Public Report	Verichains Lab

Table 3. Report versions history