KingSpeed Security Analysis

1. Overview Abstract

In this report, we consider the security of the KingSpeed token. Our main task is to find and describe security issues in the smart contracts of the platform to the team.

1.1 Limitations and use of the report

Broadly speaking, the assessments can not uncover all vulnerabilities of the given smart contracts, thus, it is not guaranteed that the system is secured even if no vulnerabilities are found. The focus of the assessments was limited to given smart contracts, other contracts were excluded (including external libraries or third party codes).

1.2 Summary

We have found **2** high severity issues, **1** medium severity issue and **1** low severity issue.

1.3 Recommendations

We recommend the team to fix all issues, as well as test coverage to ensure the security of the contracts.

2. Assessment Overview

2.1 Scope of the audit

Source codes for the audit were initially taken from the deployed contract address 0xceb03ce16dc28b6c8c776fd2cd050b883019a9aa. The team has taken the implementation of the CAKE token as references.

After the initial report for the source codes above, the team has fixed some of the findings.

The source codes for the final report is taken from the deployed contract address 0x3AC0F8CEcc1Fb0Ee6C2017A072D52E85B00c6694

3. System Overview

KingSpeed Crystal (KSC) token is the utility token of the KingSpeed ecosystem. The token is used in a number of ways:

- To purchase certain types of Collectibles.
- To use as fuel to open loot boxes.
- To stake and participate in governance votes, give users the incentives to hold the token.
- To rent NFT cars.
- To use to participate in a prediction game.
- To use as a reward for completing certain actions or achievements in game or winning a race.
- To use as a fee to compete in the 1-1 racing game.

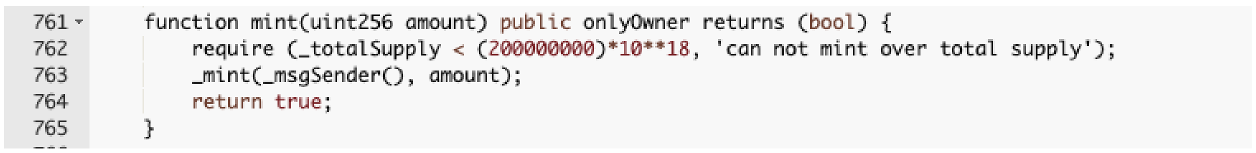
More information can be found on their docs website.

4. Findings

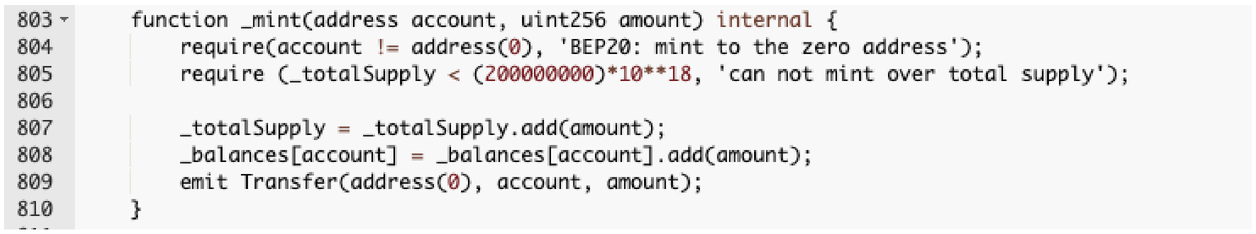
We have found **2** high severity issues, **1** medium severity issue and **1** low severity issue.

[High][Partial-Fixed] 4.1 Can mint more than max total supply

There are 2 checks in the _mint and mint function to avoid minting more than total supply.



```
    function mint(uint256 amount) public onlyOwner returns (bool) {
        require (_totalSupply < (200000000)*10**18, 'can not mint over total supply');
        _mint(_msgSender(), amount);
        return true;
    }
```



```
    function _mint(address account, uint256 amount) internal {
        require(account != address(0), 'BEP20: mint to the zero address');
        require (_totalSupply < (200000000)*10**18, 'can not mint over total supply');

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }
```

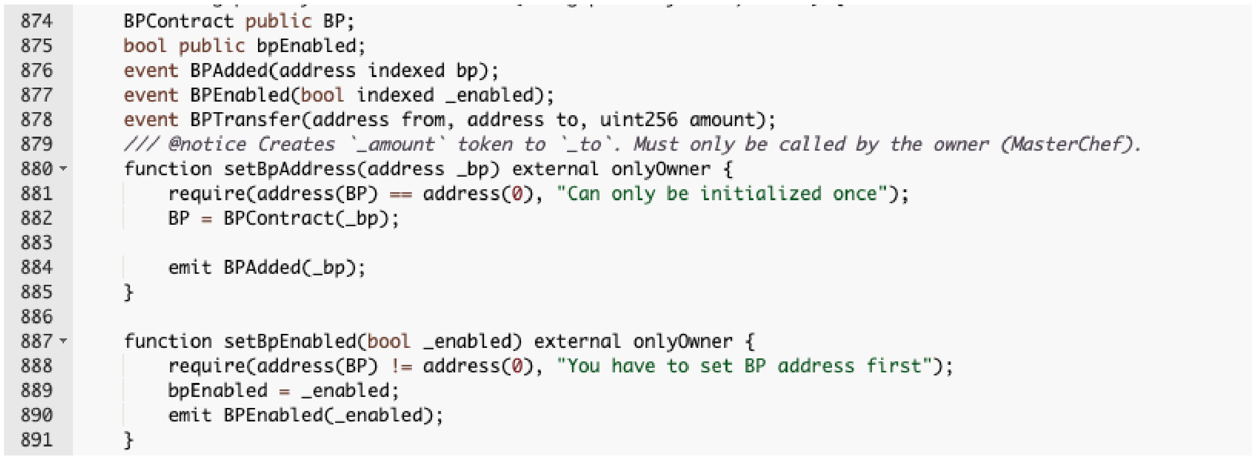
These checks don't work as the **_totalSupply** value is updated after the required condition, thus, the owner can still mint more than **200,000,000 * 10**18** tokens.

Moreover, it is redundant to check in the **mint** function as it is calling the **_mint** function.

Comment: Team has fixed this by minting max total supply of 200M tokens from the constructor. However the team decided to keep redundant mint functions.

[High][Fixed] 4.2 No implementation for bot protection

The contract has a Bot Protection address and flag to enable/disable the bot protection mechanism.



```
    BPContract public BP;
    bool public bpEnabled;
    event BPAdded(address indexed bp);
    event BPEnabled(bool indexed _enabled);
    event BPTransfer(address from, address to, uint256 amount);
    /// @notice Creates `_amount` token to `_to`. Must only be called by the owner (MasterChef).
    function setBpAddress(address _bp) external onlyOwner {
        require(address(BP) == address(0), "Can only be initialized once");
        BP = BPContract(_bp);

        emit BPAdded(_bp);
    }

    function setBpEnabled(bool _enabled) external onlyOwner {
        require(address(BP) != address(0), "You have to set BP address first");
        bpEnabled = _enabled;
        emit BPEnabled(_enabled);
    }
```

However, there is no implementation for bot protection in the source codes (more specifically, in the transfer function) as the team has mentioned.

Comment: This issue has been adjusted in the latest code.

[Medium] [No code changes] 4.3 Wrongly implement of votes mechanism

The contract did not update the delegate state whenever a transfer occurred. This gives the wrong result whenever a user delegates their vote then transfers the token.

Comment: Team is aware of the issue and decided not to make any changes.

[Low] [No code changes] 4.4 High gas consumption for all interactions when using checkpoint

The team is using checkpoint mechanism for governance votes purpose, however, it is known that it costs a high amount of gas consumption for all interactions with the token.

Comment: Team is aware of the issue and decided not to make any changes.