freshcoins

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

SUPERSTAKE

$SUPERSTAKE

14/01/2023

TOKEN OVERVIEW

Fees

• Buy fees:	7%
• Sell fees:	7%

Fees privileges

- Can change fees up to 15%

Ownership

- Owned

Minting

- No mint function

Max Tx Amount / Max Wallet Amount

- Can’t change max tx amount or max wallet amount

Blacklist

- Blacklist function not detected

Other privileges

- Can exclude / include from fees

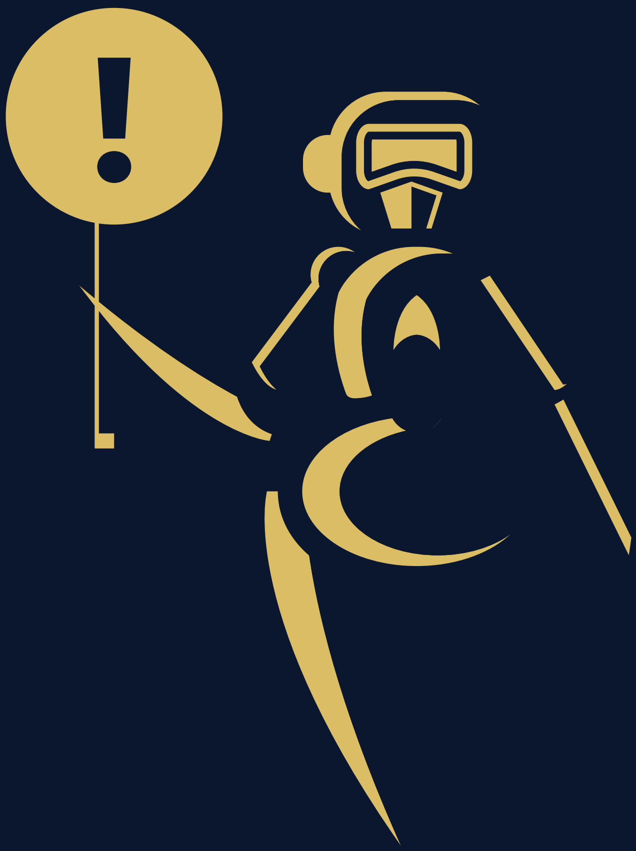
TABLE OF CONTENTS

DISCLAIMER

The information provided on this analysis document is only for general information and should not be used as a reason to invest.

FreshCoins Team will take no payment for manipulating the results of this audit.

The score and the result will stay on this project page information on our website https://freshcoins.io
FreshCoins Team does not guarantees that a project will not sell off team supply, or any other scam strategy (RUG or Honeypot etc)



INTRODUCTION

FreshCoins (Consultant) was contracted by SUPERSTAKE (Customer) to conduct a Smart Contract Code Review and Security Analysis.

0xad5452bE97e257f7312AC7C681407a8B49Ff1408

Network: Binance Smart Chain (BSC)

This report presents the findings of the security assessment of Customer's smart contract and its code review conducted on 14/01/2023

AUDIT OVERVIEW

91

Security Score

89

Static Scan

Automatic scanning for common vulnerabilities

91

ERC Scan

Automatic checks for ERC's conformance

0	High
3	Medium
0	Low
0	Optimizations
0	Informational

No.	Issue description	Checking Status
1	Compiler Errors / Warnings	Passed
2	Reentrancy and Cross-function	Passed
3	Front running	Passed
4	Timestamp dependence	Passed
5	Integer Overflow and Underflow	Passed
6	Reverted DoS	Passed
7	DoS with block gas limit	Low
8	Methods execution permissions	Passed
9	Exchange rate impact	Passed
10	Malicious Event	Passed
11	Scoping and Declarations	Passed
12	Uninitialized storage pointers	Passed
13	Design Logic	Passed
14	Safe Zeppelin module	Passed

OWNER PRIVILEGES

- ● Contract owner can't mint tokens after initial contract deploy

- ● Contract owner can't exclude an address from transactions

- ● Contract owner can exclude/include wallet from tax

```
function manage_FeeExempt(address[] calldata addresses, bool status)
        external
        onlyOwner
    {
        require(
            addresses.length < 501,
            "GAS Error: max limit is 500 addresses"
        );
        for (uint256 i = 0; i < addresses.length; ++i) {
            isFeeExempt[addresses[i]] = status;
            emit Wallet_feeExempt(addresses[i], status);
        }
}
```

- ● Contract owner can send tokens to multiple wallets

```
function multiTransfer(
        address[] calldata addresses,
        uint256[] calldata tokens
    ) external {
        require(isFeeExempt[msg.sender]);
        address from = msg.sender;
        require(
            addresses.length < 501,
            "GAS Error: max limit is 500 addresses"
        );
        require(
            addresses.length == tokens.length,
            "Mismatch between address and token count"
        );
        uint256 SCCC = 0;
        for (uint256 i = 0; i < addresses.length; i++) {
            SCCC = SCCC + tokens[i];
        }
        require(balanceOf[from] >= SCCC, "Not enough tokens in wallet");

        for (uint256 i = 0; i < addresses.length; i++) {
            _basicTransfer(from, addresses[i], tokens[i]);
        }
}
```

Contract owner has to call tradingEnable() function to enable trade

```
function tradingEnable() external onlyOwner {
    require(!tradingOpen, "Trading already open"); //trade only can change one time
    tradingOpen = true;
    launchedAt = block.timestamp;
    emit config_TradingStatus(tradingOpen);
}
```

Contract owner can change fees up to 15%

```
function setFees(uint256 _liquidityFee, uint256 _reflectionFee, uint256 _stakePoolFee, uint256 _burnFee,
uint256 _marketingFee, uint256 _sellLiquidityFee, uint256 _sellReflectionFee, uint256 _sellStakeoolFee,
uint256 _sellBurnFee, uint256 _sellMarketingFee    ) external onlyOwner {
    liquidityFee = _liquidityFee;
    marketingFee = _marketingFee;
    reflectionFee = _reflectionFee;
    stakePoolFee = _stakePoolFee;
    burnFee = _burnFee;

    totalBuyFee =
      _liquidityFee +
      _marketingFee +
      _stakePoolFee +
      _reflectionFee +
      _burnFee;

    sellLiquidityFee = _sellLiquidityFee;
    sellReflectionFee = _sellReflectionFee;
    sellStakePoolFee = _sellStakePoolFee;
    sellBurnFee = _sellBurnFee;
    sellMarketingFee = _sellMarketingFee;

    totalSellFee =
      _sellLiquidityFee +
      _sellReflectionFee +
      _sellStakePoolFee +
      _sellBurnFee +
      _sellMarketingFee;
    update_fees();
}

function update_fees() internal {
    require(
      (totalBuyFee + totalSellFee * (200)) / 100 <= 150,
      "Buy+Sell tax cannot be more than 15%"
    );

    emit UpdateFee(
      uint8((totalBuyFee * 100) / 100),
      uint8((totalSellFee * 100) / 100)
    );
}
```

● Contract owner can change marketingFeeReceiver and StakePoolReceiver addresses

Default values:

marketingFeeReceiver : 0xd3ce7871a6640885fe7942afb13c45821a9bdd8f

StakePoolReceiver : 0xd3ce7871a6640885fe7942afb13c45821a9bdd8f

```
function setFeeReceivers(
        address _marketingFeeReceiver,
        address _stakePoolReceiver
    ) external onlyOwner {
        require(
            _marketingFeeReceiver != address(0),
            "Marketing fee address cannot be zero address"
        );
        require(
            _stakePoolReceiver != address(0),
            "StalePool fee address cannot be zero address"
        );
        marketingFeeReceiver = _marketingFeeReceiver;
        StakePoolReceiver = _stakePoolReceiver;
        emit Set_Wallets(marketingFeeReceiver, StakePoolReceiver);
}
```

● Contract owner can change swap settings

```
function setSwapBackSettings(bool _enabled, uint256 _amount)
        external
        onlyOwner
    {
        require(_amount < (totalSupply / 10), "Amount too high");

        swapEnabled = _enabled;
        swapThreshold = _amount;

        emit config_SwapSettings(swapThreshold, swapEnabled);
}
```

● Contract owner can disable burn

Current value: bool public burnEnabled = true;

```
function disableBurns() external onlyOwner {
        burnEnabled = false;
}
```

● Contract owner can burn tokens

```
function burn(uint256 amount) external {
        _burn(msg.sender, amount);
}
```

```
function SuperBurn(uint256 percent) public onlyOwner returns (bool) {
    require(percent <= 200, "May not nuke more than 2% of tokens in LP");
    require(block.timestamp > lastSync + 5 minutes, "Too soon");
    require(burnEnabled, "Burns are disabled");

    uint256 lp_tokens = this.balanceOf(address(pairContract));
    uint256 lp_burn = (lp_tokens * percent) / 10_000;

    if (lp_burn > 0) {
      _burn(address(pairContract), lp_burn);
      pairContract.sync();
      return true;
    }

    return false;
}

function _burn(address account, uint256 amount) internal {
    require(amount != 0);
    require(amount <= balanceOf[account]);
    balanceOf[account] = balanceOf[account] - amount;
    totalSupply = totalSupply - amount;
    emit Transfer(account, address(0), amount);
}
```

Contract owner can withdraw tokens from smart contract

Cannot withdraw native token for 1 year after launch date

```
function clearStuckToken(address tokenAddress, uint256 tokens)
    external
    onlyOwner
    returns (bool success)
  {
    require(tokenAddress != address(this), "Cannot withdraw native token");
    if (pairs[tokenAddress]) {
      //tokens from auto liquidify
      require(
        block.timestamp > launchedAt + 500 days,
        "Locked for 1 year"
      );
    }

    if (tokens == 0) {
      tokens = BEP20(tokenAddress).balanceOf(address(this));
    }

    emit clearToken(tokenAddress, tokens);

    return BEP20(tokenAddress).transfer(msg.sender, tokens);
}
```

Contract owner can transfer ownership

```
function transferOwnership(address newOwner) public onlyOwner {
      _transferOwnership(newOwner);
}

function _transferOwnership(address newOwner) internal {
      require(
         newOwner != address(0),
         "Ownable: new owner is the zero address"
      );
      emit OwnershipTransferred(_owner, newOwner);
      _owner = newOwner;
}
```

Contract owner can renounce ownership

```
function renounceOwnership() public onlyOwner {
      emit OwnershipTransferred(_owner, address(0));
      _owner = address(0);
}
```

Recommendation:

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. The risk can be prevented by temporarily locking the contract or renouncing ownership.

CONCLUSION AND ANALYSIS

Smart Contracts within the scope were manually reviewed and analyzed with static tools.

Audit report overview contains all found security vulnerabilities and other issues in the reviewed code.

Found no HIGH issues during the first review.

TOKEN DETAILS

Details

Buy fees:	7%
Sell fees:	7%
Max TX:	N/A
Max Sell:	N/A

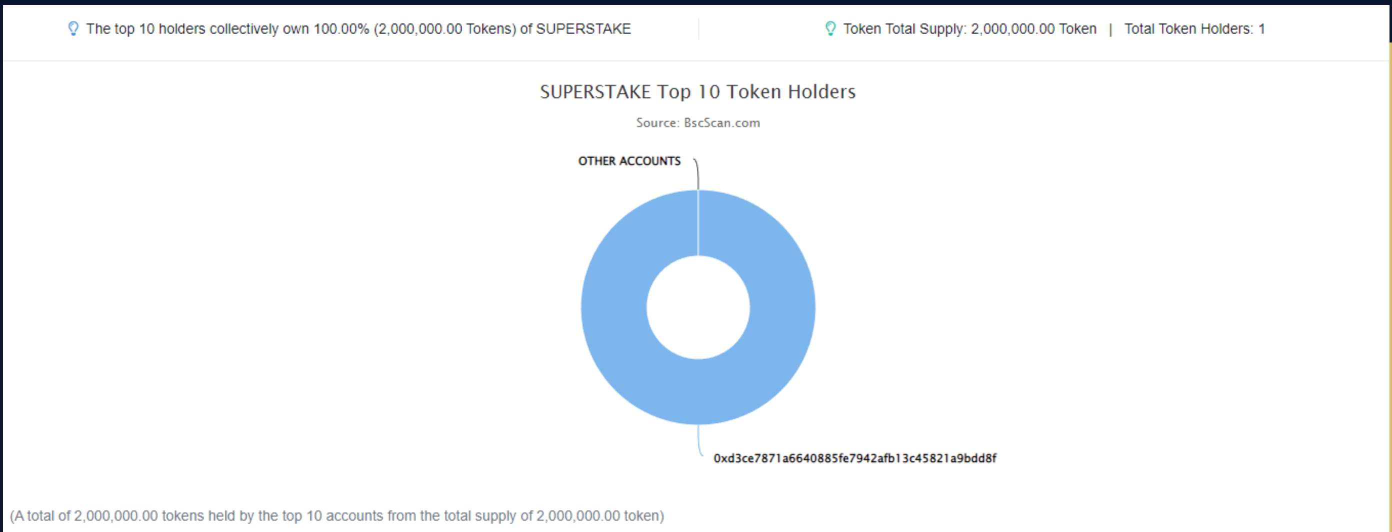
Honeypot Risk

Ownership:	Owned
Blacklist:	Not detected
Modify Max TX:	Not detected
Modify Max Sell:	Not detected
Disable Trading:	Not detected

Rug Pull Risk

Liquidity:	N/A
Holders:	Clean

SUPERSTAKE TOKEN ANALYTICS & TOP 10 TOKEN HOLDERS



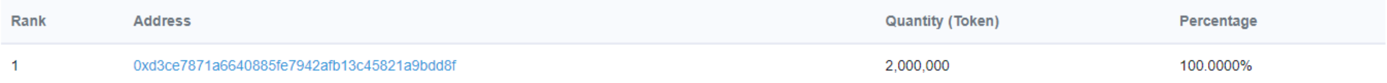
The top 10 holders collectively own 100.00% (2,000,000.00 Tokens) of SUPERSTAKE

Token Total Supply: 2,000,000.00 Token | Total Token Holders: 1

SUPERSTAKE Top 10 Token Holders

Source: BscScan.com

(A total of 2,000,000.00 tokens held by the top 10 accounts from the total supply of 2,000,000.00 token)



Rank	Address	Quantity (Token)	Percentage
1	0xd3ce7871a6640885fe7942afb13c45821a9bdd8f	2,000,000	100.0000%

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 its vulnerabilities that can lead to hacks. The audit can't guarantee the explicit security of the audited project / smart contract.