freshcoins

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Hiroki

$HIRO

31/01/2023

TOKEN OVERVIEW

Fees

- Buy fees: 0%
- Sell fees: 0%

Fees privileges

- Can change fees up to 100%

Ownership

- Owned

Minting

- Mint function detected

Max Tx Amount / Max Wallet Amount

- Can change max tx amount and max wallet amount without threshold

Blacklist

- No blacklist function

Other privileges

- Can exclude / include from fees

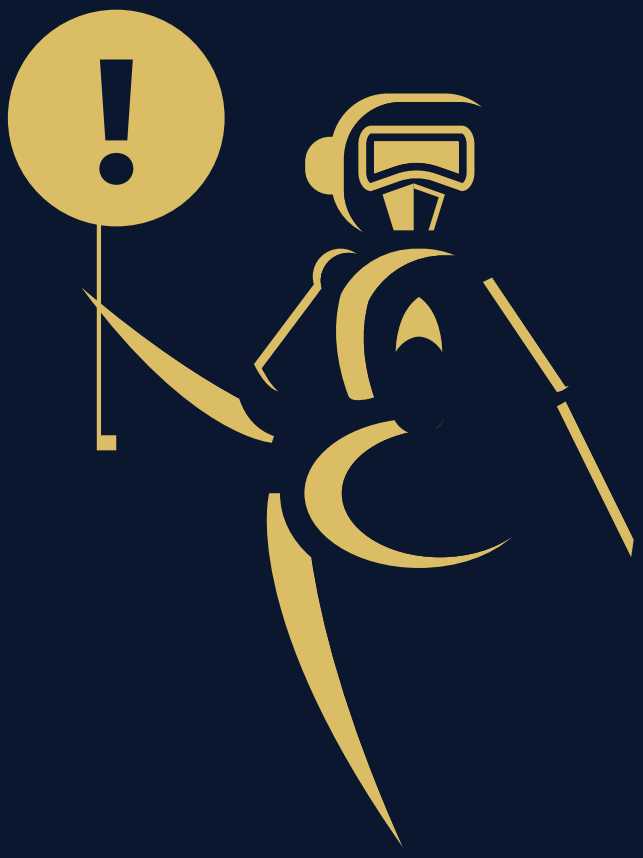
TABLE OF CONTENTS

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 Hiroki (Customer) to conduct a Smart Contract Code Review and Security Analysis.

0x64b0a8b9633697EAF6B2B2d41A4196257cc8E1dd

Network: Ethereum (ETH)

This report presents the findings of the security assessment of Customer's smart contract and its code review conducted on 31/01/2023

AUDIT OVERVIEW

61

Security Score

HIGH RISK

93

Static Scan

Automatic scanning for common vulnerabilities

72

ERC Scan

Automatic checks for ERC's conformance

4 High

4 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	Passed
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 mint tokens after initial contract deploy

```
function mint(address to, uint amount) external onlyMinter {
    _mint(to, amount);
}

function claimNewToken(uint amount) external {
    require(
      oldContractAddress != address(0),
      "Claim new token is not available!"
    );
    IERC20(oldContractAddress).transferFrom(
      msg.sender,
      address(this),
      amount
    );
    _mint(msg.sender, amount);
}
```

Contract owner can burn tokens

```
function burn(uint amount) external {
    _burn(msg.sender, amount);
    emit Burn(msg.sender, amount);
}
```

Contract owner can exclude/include wallet from tax

```
function setExcludFromFee(address to, bool _excluded) external onlyOwner {
    isExcludeFromFee[to] = _excluded;
}
```

Contract owner can exclude/include wallet from wallet limitations

```
function setExcludFromMaxWallet(address to, bool _excluded) external onlyOwner {
    isExcludeFromMaxWallet[to] = _excluded;
}
```

Contract owner can whitelist wallets

```
function setWL(address[] memory tos) external onlyOwner {
    for (uint i = 0; i < tos.length; i++) {
      isOnWL[tos[i]] = true;
    }
}
```

Contract owner can change swap settings

```
function setPresale(bool _isOnPresale) external onlyOwner {
    isOnPresale = _isOnPresale;
    if (!isOnPresale) _swapAndLiquifyEnabled = true;
}
```

Contract owner can exclude/include wallet from rewards

```
function setExcludedFromReward(address to, bool data) external onlyOwner {
        if (isExcludedFromReward[to] != data) {
            isExcludedFromReward[to] = data;
            if (data) {
                // set to exclude
                totalExcludedAmount += balanceOf(to);
            } else {
                // remove from exclude
                totalExcludedAmount -= balanceOf(to);
                uint newRewardBalance = getClaimableReward(to);
                rewardedAmount[msg.sender] += newRewardBalance;
            }
        }
    }
```

Contract owner can change max tx amount and max wallet amount limitations to 0 (without threshold)

Transfers can be disabled if _maxTxAmount value is set to 0

```
function setTxLimit(
        uint maxTxAmount,
        uint maxAmountPerWallet,
        uint numTokensSellToAddToLiquidity,
        bool swapAndLiquifyEnabled
    ) external onlyOwner {
        _maxTxAmount = maxTxAmount;
        _maxAmountPerWallet = maxAmountPerWallet;
        _numTokensSellToAddToLiquidity = numTokensSellToAddToLiquidity;
        swapAndLiquifyEnabled = swapAndLiquifyEnabled;
    }
```

Contract owner can change marketing, nft, team and reserve addresses

Current values:

marketing : 0x502fEe1Ae245DeD7b2C2cE382bf4A71C919a4C66

nft: 0x859C851F19DcB516BcdC42Eec87C1b2bE0F91562

team: 0x8fF8657d9def00fa0d20aAeDb2E92834A2Ec481D

reserve: 0xfa9edbA55314dD71F8A31ec31c684bFFA4B7cd0b

```
function setFeeWallets(FeeWallet calldata _feeWallets) external onlyOwner {
        feeWallets.marketing = _feeWallets.marketing;
        feeWallets.nft = _feeWallets.nft;
        feeWallets.team = _feeWallets.team;
        feeWallets.reserve = _feeWallets.reserve;
    }
```

Contract owner can withdraw stuck tokens from smart contract

```
function claimstuckedToken(address token, uint amount) external onlyOwner {
    if (token == address(0)) payable(msg.sender).transfer(amount);
    else IERC20(token).transfer(msg.sender, amount);
}
```

Contract owner can change fees up to 100%

```
function setFees(Fee calldata _sellFees, Fee calldata _buyFees)
    external
    onlyOwner
  {
    sellFees.marketing_fee = _sellFees.marketing_fee;
    sellFees.NFT_fee = _sellFees.NFT_fee;
    sellFees.team_fee = _sellFees.team_fee;
    sellFees.reserve_fee = _sellFees.reserve_fee;
    sellFees.reward_fee = _sellFees.reward_fee;
    sellFees.auto_lp = _sellFees.auto_lp;

    buyFees.marketing_fee = _buyFees.marketing_fee;
    buyFees.NFT_fee = _buyFees.NFT_fee;
    buyFees.team_fee = _buyFees.team_fee;
    buyFees.reserve_fee = _buyFees.reserve_fee;
    buyFees.reward_fee = _buyFees.reward_fee;
    buyFees.auto_lp = _buyFees.auto_lp;
}
```

Contract owner can allow any wallet access to mint function

```
function setMinter(address to, bool isMinter) external onlyOwner {
    isMinters[to] = isMinter;
}
```

Contract owner can renounce ownership

```
function renounceOwnership() public virtual onlyOwner {
    _transferOwnership(address(0));
}
```

Contract owner can transfer ownership

```
function transferOwnership(address newOwner) public virtual onlyOwner {
    require(
      newOwner != address(0),
      "Ownable: new owner is the zero address"
    );
    _transferOwnership(newOwner);
}
```

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 4 HIGH issues during the first review.

TOKEN DETAILS

Details

Buy fees:	0%
Sell fees:	0%
Max TX:	1,000,000
Max Sell:	N/A

Honeypot Risk

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

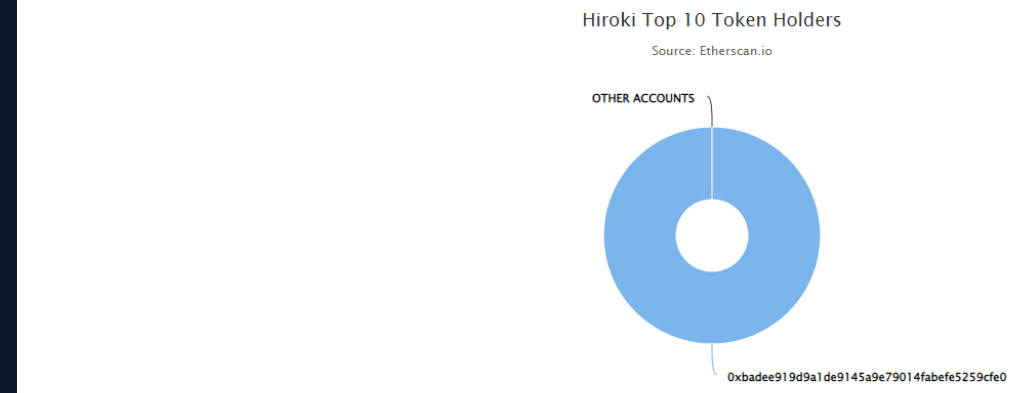
Others

Liquidity:	N/A
Holders:	Clean

HIRO TOKEN ANALYTICS & TOP 10 TOKEN HOLDERS

The top 10 holders collectively own 100.00% (100,000,000.00 Tokens) of Hiroki

Token Total Supply: 100,000,000.00 Token | Total Token Holders: 1



(A total of 100,000,000.00 tokens held by the top 10 accounts from the total supply of 100,000,000.00 token)

Rank	Address	Quantity (Token)	Percentage
1	0xbadee919d9a1de9145a9e79014fabefe5259cfe0	100,000,000	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.