freshcoins

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Octus TIGER

$octTIGER

16/10/2022

TOKEN OVERVIEW

Fees

• Buy fees:	N/A
• Sell fees:	N/A

Fees privileges

• Can’t set / change fees

Ownership

• Owned

Minting

• Mint function detected

Max Tx Amount / Max Wallet Amount

• Can’t set / change max tx amount and / or wallet limitations

Blacklist

• No blacklist function

Other privileges

• Can burn tokens from any wallet address

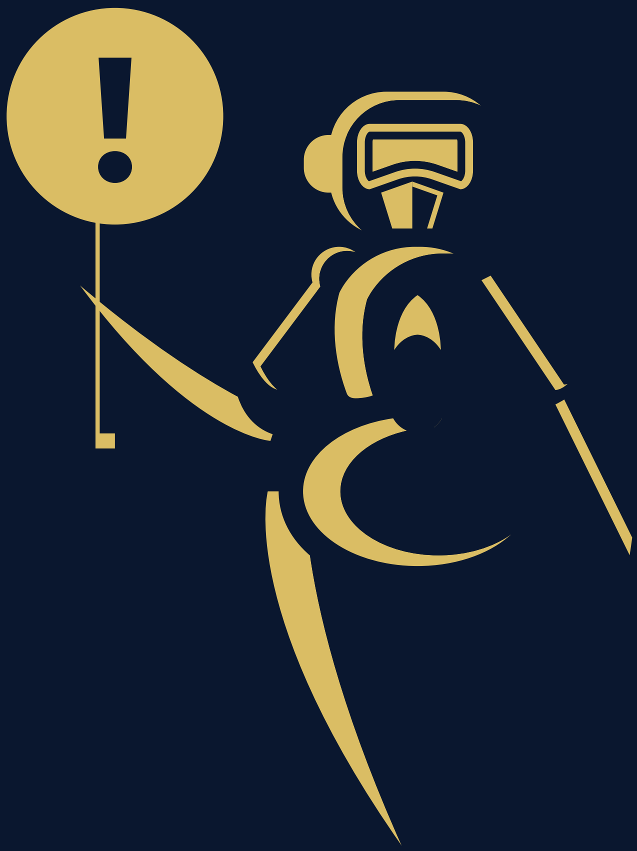
TABLE OF CONTENTS

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

0x07dbEF0F356623168e6279788507Bd98dd9D6304

Network: Ethereum (ETH)

This report presents the findings of the security assessment of Customer's smart contract and its code review conducted on 16/10/2022

AUDIT OVERVIEW

42

Security Score
AUDIT: FAILED

92 **Static Scan**
Automatic scanning for common vulnerabilities

40 **ERC Scan**
Automatic checks for ERC's conformance

2	High
0	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 account,
      uint amount
   ) external override onlyOwner {
      _mint(account, amount);
}

function _mint(address account, uint256 amount) internal virtual {
      require(account != address(0), "ERC20: mint to the zero address");

      _beforeTokenTransfer(address(0), account, amount);

      _totalSupply += amount;
      _balances[account] += amount;
      emit Transfer(address(0), account, amount);

      _afterTokenTransfer(address(0), account, amount);
}
```

● Contract owner can burn tokens from any wallet address

```
function burn(
      address account,
      uint amount
   ) external override onlyOwner {
      _burn(account, amount);
}

function _burn(address account, uint256 amount) internal virtual {
      require(account != address(0), "ERC20: burn from the zero address");

      _beforeTokenTransfer(account, address(0), amount);

      uint256 accountBalance = _balances[account];
      require(accountBalance >= amount, "ERC20: burn amount exceeds balance");
   unchecked {
      _balances[account] = accountBalance - amount;
   }
      _totalSupply -= amount;

      emit Transfer(account, address(0), amount);

      _afterTokenTransfer(account, address(0), amount);
}
```

● 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 2 HIGH issues during the first review.

TOKEN DETAILS

Details

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

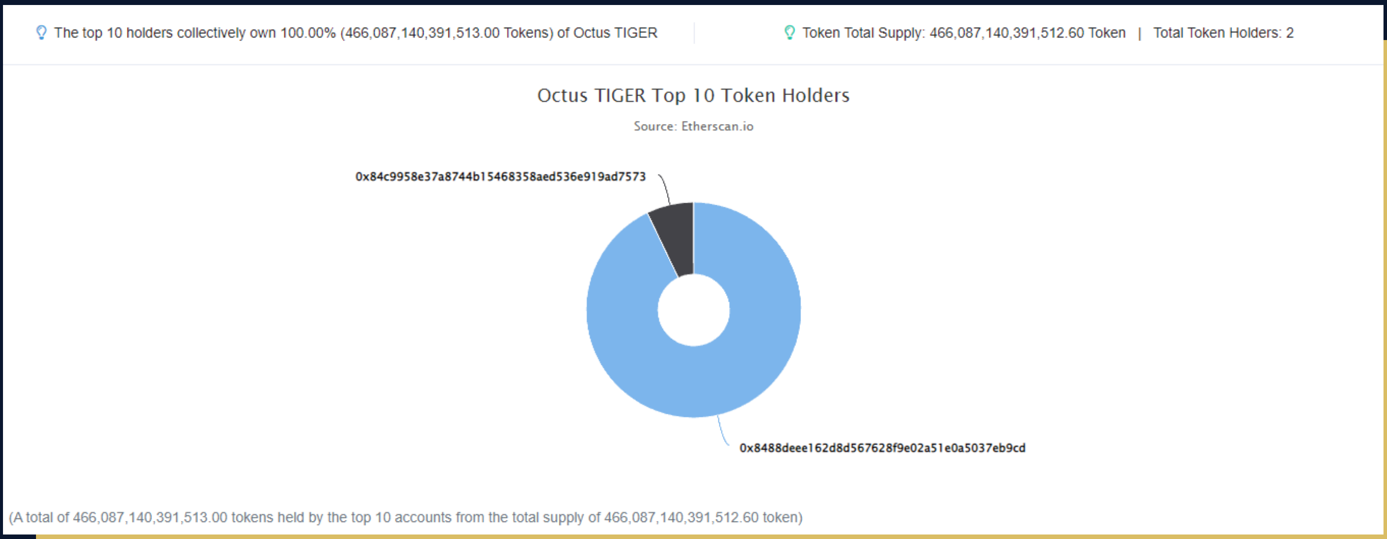
Honeypot Risk

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

Others

Liquidity:	N/A
Holders:	Clean

OCTUS TIGER TOKEN ANALYTICS & TOP 10 TOKEN HOLDERS



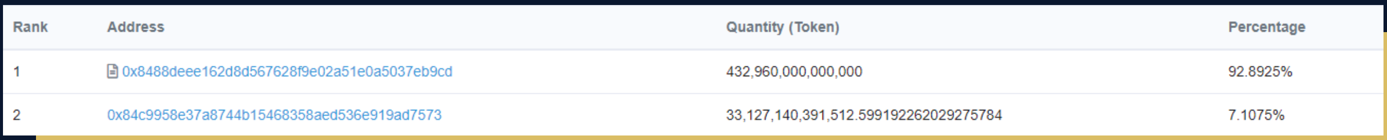
The top 10 holders collectively own 100.00% (466,087,140,391,513.00 Tokens) of Octus TIGER

Token Total Supply: 466,087,140,391,512.60 Token | Total Token Holders: 2

Octus TIGER Top 10 Token Holders

Source: Etherscan.io

(A total of 466,087,140,391,513.00 tokens held by the top 10 accounts from the total supply of 466,087,140,391,512.60 token)



Rank	Address	Quantity (Token)	Percentage
1	0x8488deee162d8d567628f9e02a51e0a5037eb9cd	432,960,000,000,000	92.8925%
2	0x84c9958e37a8744b15468358aed536e919ad7573	33,127,140,391,512.599192262029275784	7.1075%

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.