智能合约审计报告

降维安全
JohnWick Sec

守 护 价 值 互 联 网

降 维 安 全 实 验 室

John Wick Security Lab received the BxC (company/team) BonusCloud Token (BxC) project smart contract code audit requirements on 2018/09/12.

Project Name：BonusCloud Token (BxC)

Smart Contract Address：

https://etherscan.io/address/0xdeCF7Be29F8832E9C2Ddf0388c9778B8Ba76af43#code

Audit No.：201809008

Audit Category and Result:

Audit category	Subcategory	Result
Overflow	-	Pass
Race condition	-	Pass
Access control	-	Pass
Denial of service	-	Pass
Gas optimization	-	Pass
Programming	Compiler version	Pass
	Random number generation	Pass
	Hardcoded address	Pass
	Use of fallback function	Pass
	Internal function call bypass	Pass
	Logic error	Pass
	Fake charge	Pass
	Malicious event	Pass
Special Service	Code format normalization	Pass
	Business risk audit	Pass
	Fuzzy test	Pass

(Other unknown security vulnerabilities and Ethereum design flaws are not included in this audit responsibility)

Audit Result：PASS

Audit Date：20180912

Auditor：John Wick Security Lab

（Disclaimer: The John Wick Security Lab issues this report based on the facts that have

occurred or existed before the issuance of this report, and assumes corresponding responsibility in this regard. For the facts that occur or exist after the issuance of this report, the John Wick Security Lab cannot judge the security status of its smart contracts and does not assume any responsibility for it. The safety audit analysis and other contents of this report are based on the relevant materials and documents provided by the information provider to the John Wick Security Lab when the report is issued (referred to as the information provided). The John Wick Security Lab assumes that there is no missing, falsified, deleted, or concealed information provided. If the information provided is missing, falsified, deleted, concealed, or the information provider's response is inconsistent with the actual situation, the John Wick Security Lab shall not bear any responsibility for the resulting loss and adverse effects.）

Audit Details:

//JohnWick: Specifying to use ver. 0.4.24 which is the latest version of the solidity compiler avoids security issues caused by compiler bugs, which is in line with best practices.

//JohnWick: Using the SafeMath library avoids potential integer overflow issues, which is in line with best practices.

// JohnWick: Using increaseApproval(address _spender, uint256 _addedValue) and decreaseApproval(address _spender, uint256 _subtractedValue) implements atomic modification the ‘allowance’, which is in line with best practices.

Smart Contract Source Code:

```
pragma solidity ^0.4.24;

/**
 * @dev Library that helps prevent integer overflows and underflows,
 * inspired by https://github.com/OpenZeppelin/zeppelin-solidity
 */
library SafeMath {
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a);

        return c;
    }

    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
```

```
        require(b <= a);
        uint256 c = a - b;

        return c;
    }

    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }
        uint256 c = a * b;
        require(c / a == b);

        return c;
    }

    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0);
        uint256 c = a / b;

        return c;
    }
}

/**
 * @title HasOwner
 *
 * @dev Allows for exclusive access to certain functionality.
 */
contract HasOwner {
    // Current owner.
    address public owner;

    // Conditionally the new owner.
    address public newOwner;

    /**
     * @dev The constructor.
     *
     * @param _owner The address of the owner.
     */
    constructor(address _owner) internal {
        owner = _owner;
    }
```

```
    /**
     * @dev Access control modifier that allows only the current owner to call
the function.
     */
    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }

    /**
     * @dev The event is fired when the current owner is changed.
     *
     * @param _oldOwner The address of the previous owner.
     * @param _newOwner The address of the new owner.
     */
    event  OwnershipTransfer(address  indexed  _oldOwner,  address  indexed
_newOwner);

    /**
     * @dev Transfering the ownership is a two-step process, as we prepare
     * for the transfer by setting `newOwner` and requiring `newOwner` to accept
     * the transfer. This prevents accidental lock-out if something goes wrong
     * when passing the `newOwner` address.
     *
     * @param _newOwner The address of the proposed new owner.
     */
    function transferOwnership(address _newOwner) public onlyOwner {
        newOwner = _newOwner;
    }

    /**
     * @dev The `newOwner` finishes the ownership transfer process by accepting
the
     * ownership.
     */
    function acceptOwnership() public {
        require(msg.sender == newOwner);

        emit OwnershipTransfer(owner, newOwner);

        owner = newOwner;
    }
}

/**
```

```
 * @dev The standard ERC20 Token interface.
 */
contract ERC20TokenInterface {
    uint256 public totalSupply;  /* shorthand for public function and a property
*/
    event Transfer(address indexed _from, address indexed _to, uint256 _value);
    event Approval(address indexed _owner, address indexed _spender, uint256
_value);
    function balanceOf(address _owner) public constant returns (uint256
balance);
    function transfer(address _to, uint256 _value) public returns (bool
success);
    function transferFrom(address _from, address _to, uint256 _value) public
returns (bool success);
    function approve(address _spender, uint256 _value) public returns (bool
success);
    function allowance(address _owner, address _spender) public constant
returns (uint256 remaining);
}

/**
 * @title ERC20Token
 *
 * @dev Implements the operations declared in the `ERC20TokenInterface`.
 */
contract ERC20Token is ERC20TokenInterface {
    using SafeMath for uint256;

    // Token account balances.
    mapping (address => uint256) balances;

    // Delegated number of tokens to transfer.
    mapping (address => mapping (address => uint256)) allowed;

    /**
     * @dev Checks the balance of a certain address.
     *
     * @param _account The address which's balance will be checked.
     *
     * @return Returns the balance of the `_account` address.
     */
    function balanceOf(address _account) public constant returns (uint256
balance) {
        return balances[_account];
    }
```

```
    /**
     * @dev Transfers tokens from one address to another.
     *
     * @param _to The target address to which the `_value` number of tokens will
be sent.
     * @param _value The number of tokens to send.
     *
     * @return Whether the transfer was successful or not.
     */
    function transfer(address _to, uint256 _value) public returns (bool success)
{
        require(_to != address(0));
        require(_value <= balances[msg.sender]);
        require(_value > 0);

        balances[msg.sender] = balances[msg.sender].sub(_value);
        balances[_to] = balances[_to].add(_value);

        emit Transfer(msg.sender, _to, _value);

        return true;
    }

    /**
     * @dev Send `_value` tokens to `_to` from `_from` if `_from` has approved
the process.
     *
     * @param _from The address of the sender.
     * @param _to The address of the recipient.
     * @param _value The number of tokens to be transferred.
     *
     * @return Whether the transfer was successful or not.
     */
    function transferFrom(address _from, address _to, uint256 _value) public
returns (bool success) {
        require(_value <= balances[_from]);
        require(_value <= allowed[_from][msg.sender]);
        require(_value > 0);
        require(_to != address(0));

        balances[_from] = balances[_from].sub(_value);
        balances[_to] = balances[_to].add(_value);
        allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
```

```
        emit Transfer(_from, _to, _value);

        return true;
    }

    /**
     * @dev Allows another contract to spend some tokens on your behalf.
     *
     * @param _spender The address of the account which will be approved for
transfer of tokens.
     * @param _value The number of tokens to be approved for transfer.
     *
     * @return Whether the approval was successful or not.
     */
    function approve(address _spender, uint256 _value) public returns (bool
success) {
        allowed[msg.sender][_spender] = _value;

        emit Approval(msg.sender, _spender, _value);

        return true;
    }

    /**
     * @dev Increase the amount of tokens that an owner allowed to a spender.
     * approve should be called when allowed[_spender] == 0. To increment
     * allowed value is better to use this function to avoid 2 calls (and wait
until
     * the first transaction is mined)
     * From MonolithDAO Token.sol
     *
     * @param _spender The address which will spend the funds.
     * @param _addedValue The amount of tokens to increase the allowance by.
     */
    function increaseApproval(address _spender, uint256 _addedValue) public
returns (bool) {
        allowed[msg.sender][_spender]                                        =
(allowed[msg.sender][_spender].add(_addedValue));

        emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);

        return true;
    }

    /**
```

```
     * @dev Decrease the amount of tokens that an owner allowed to a spender.
     * approve should be called when allowed[_spender] == 0. To decrement
     * allowed value is better to use this function to avoid 2 calls (and wait
until
     * the first transaction is mined)
     * From MonolithDAO Token.sol
     *
     * @param _spender The address which will spend the funds.
     * @param _subtractedValue The amount of tokens to decrease the allowance
by.
     */
    function decreaseApproval(address _spender, uint256 _subtractedValue)
public returns (bool) {
        uint256 oldValue = allowed[msg.sender][_spender];
        if (_subtractedValue >= oldValue) {
            allowed[msg.sender][_spender] = 0;
        } else {
            allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
        }

        emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
        return true;
    }

    /**
     * @dev Shows the number of tokens approved by `_owner` that are allowed
to be transferred by `_spender`.
     *
     * @param _owner The account which allowed the transfer.
     * @param _spender The account which will spend the tokens.
     *
     * @return The number of tokens to be transferred.
     */
    function allowance(address _owner, address _spender) public constant
returns (uint256 remaining) {
        return allowed[_owner][_spender];
    }

    /**
     * Don't accept ETH
     */
    function () public payable {
        revert();
    }
}
```

```
/**
 * @title Freezable
 * @dev This trait allows to freeze the transactions in a Token
 */
contract Freezable is HasOwner {
    bool public frozen = false;

    /**
     * @dev Modifier makes methods callable only when the contract is not frozen.
     */
    modifier requireNotFrozen() {
        require(!frozen);
        _;
    }

    /**
     * @dev Allows the owner to "freeze" the contract.
     */
    function freeze() onlyOwner public {
        frozen = true;
    }

    /**
     * @dev Allows the owner to "unfreeze" the contract.
     */
    function unfreeze() onlyOwner public {
        frozen = false;
    }
}

/**
 * @title FreezableERC20Token
 *
 * @dev Extends ERC20Token and adds ability to freeze all transfers of tokens.
 */
contract FreezableERC20Token is ERC20Token, Freezable {
    /**
     * @dev Overrides the original ERC20Token implementation by adding
whenNotFrozen modifier.
     *
     * @param _to The target address to which the `_value` number of tokens will
be sent.
     * @param _value The number of tokens to send.
     *
```

```
     * @return Whether the transfer was successful or not.
     */
    function transfer(address _to, uint _value) public requireNotFrozen returns
(bool success) {
        return super.transfer(_to, _value);
    }

    /**
     * @dev Send `_value` tokens to `_to` from `_from` if `_from` has approved
the process.
     *
     * @param _from The address of the sender.
     * @param _to The address of the recipient.
     * @param _value The number of tokens to be transferred.
     *
     * @return Whether the transfer was successful or not.
     */
    function transferFrom(address _from, address _to, uint _value) public
requireNotFrozen returns (bool success) {
        return super.transferFrom(_from, _to, _value);
    }

    /**
     * @dev Allows another contract to spend some tokens on your behalf.
     *
     * @param _spender The address of the account which will be approved for
transfer of tokens.
     * @param _value The number of tokens to be approved for transfer.
     *
     * @return Whether the approval was successful or not.
     */
    function approve(address _spender, uint _value) public requireNotFrozen
returns (bool success) {
        return super.approve(_spender, _value);
    }

    function increaseApproval(address _spender, uint256 _addedValue) public
requireNotFrozen returns (bool) {
        return super.increaseApproval(_spender, _addedValue);
    }

    function decreaseApproval(address _spender, uint256 _subtractedValue)
public requireNotFrozen returns (bool) {
        return super.decreaseApproval(_spender, _subtractedValue);
    }
```

```
}

/**
 * @title BonusCloudTokenConfig
 *
 * @dev The static configuration for the Bonus Cloud Token.
 */
contract BonusCloudTokenConfig {
    // The name of the token.
    string constant NAME = "BonusCloud Token";

    // The symbol of the token.
    string constant SYMBOL = "BxC";

    // The number of decimals for the token.
    uint8 constant DECIMALS = 18;

    // Decimal factor for multiplication purposes.
    uint256 constant DECIMALS_FACTOR = 10 ** uint(DECIMALS);

    // TotalSupply
    uint256 constant TOTAL_SUPPLY = 7000000000 * DECIMALS_FACTOR;
}

/**
 * @title Bonus Cloud Token
 *
 * @dev A standard token implementation of the ERC20 token standard with added
 *      HasOwner trait and initialized using the configuration constants.
 */
contract     BonusCloudToken     is     BonusCloudTokenConfig,     HasOwner,
FreezableERC20Token {
    // The name of the token.
    string public name;

    // The symbol for the token.
    string public symbol;

    // The decimals of the token.
    uint8 public decimals;

    /**
     * @dev The constructor.
     *
     */
```

```
    constructor() public HasOwner(msg.sender) {
        name = NAME;
        symbol = SYMBOL;
        decimals = DECIMALS;
        totalSupply = TOTAL_SUPPLY;
        balances[owner] = TOTAL_SUPPLY;
    }
}
```