# 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

降维安全实验室于 2018 年 9 月 12 日 收到 BxC（公司/团队）BonusCloud Token (BxC) 项目智能合约源代码安全审计需求。

项目名称：BonusCloud Token (BxC)

合约地址：

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

审计编号：201809008

审计项目及结果：

(其他未知安全漏洞和以太坊设计缺陷不包含在本次审计责任范围内)

| 审计大类 | 审计子类 | 审计结果（通过或未通过） |
|---|---|---|
| 溢出审计 | - | 通过 |
| 条件竞争 | - | 通过 |
| 访问控制 | - | 通过 |
| 拒绝服务 | - | 通过 |
| Gas 优化 | - | 通过 |
| 程序设计 | 编译器版本 | 通过 |
|  | 随机数生成 | 通过 |
|  | 硬编码地址审计 | 通过 |
|  | 回退函数使用 | 通过 |
|  | 内部函数调用绕过 | 通过 |
|  | 其他显性逻辑错误 | 通过 |
|  | “假充值” | 通过 |
|  | 恶意 Event 审计 | 通过 |
| 特色服务 | 代码格式规范化 | 通过 |
|  | 业务风险审计 | 通过 |
|  | 模糊测试结果 | 通过 |

审计结果：通过

审计日期：20180912

审计团队：降维安全实验室

（声明：降维安全实验室依据本报告出具前已经发生或存在的事实出具本报告，并就此承担相应责任。对

于本报告出具后，发生或存在的事实，降维安全实验室无法判断其智能合约安全状况，亦不对此承担任何责任。本报告所做的安全审计分析及其他内容，基于信息提供者截止本报告出具时向降维安全实验室提供的相关材料和文件（简称已提供资料）。降维安全实验室假设：已提供资料不存在缺失、篡改、删减、隐瞒的情况。如已提供资料存在信息缺失、被篡改、被删减、被隐瞒的情况，或资料提供者反应的情况与实际情况不符的，降维安全实验室对由此导致的损失和不利影响不承担任何责任。）

**审计详情:**

//JohnWick: 指定使用 0.4.24 版本(截至审计时最新版本)的编译器,避免了编译器 bug 导致的安全问题,符合最佳实践.

//JohnWick: 使用了 SafeMath 函数库来避免潜在的整数溢出问题,符合最佳实践.

//JohnWick: 使用了 increaseApproval(address _spender, uint256 _addedValue) 和 decreaseApproval(address _spender, uint256 _subtractedValue)实现原子化修改限额 allowance,符合最佳实践.

**审计源码:**

```
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;
    }
}
```