智能合约审计报告

守护价值互联网

北京互联共识科技有限公司

二零一八年七月

降维安全实验室于 2018 年 7 月 18 日 收到 NRC (R)（公司/团队）的 NRC (R) 项目智能合约源代码安全审计需求。

项目名称：NRC (R)

合约地址：

https://etherscan.io/address/0x7D8b9F24320Dab5369144Eb46927667f4a58dC49#code

审计编号：201807125

审计项目及结果:

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	未通过
特色服务	代码格式规范化	通过
	模糊测试结果	通过

(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)

审计结果：通过【Owner 可冻结任意用户账户，有一定业务风险】

审计日期：20180718

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。）

审计详情如下：

//John Wick:使用 SafeMath 函数库，符合推荐做法。

//John Wlck:使用回退函数，符合推荐做法。

//John Wick:Owner 权限较高【中危】,Owner 用户可以使用 freezeAccount 函数冻结任意用户地址，包括交易所地址。

//John Wick:代码逻辑错误【中危】，本合约暂停时，转账函数仍可使用。

```
pragma solidity ^ 0.4.18;


/**
 * @title Owned
 * @dev The Owned contract has an owner address, and provides basic authorization
control
 */
contract Owned {
    address public owner;
```

```
    /*Set owner of the contract*/
    function Owned() public {
        owner = msg.sender;
    }

    /*only owner can be modifier*/
    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }
}

/**
 * @title Pausable
 * @dev Base contract which allows children to implement an emergency stop mechanism.
 */
contract Pausable is Owned {
  event Pause();
  event Unpause();
```

```
bool public paused = false;




/**
 * @dev Modifier to make a function callable only when the contract is not paused.
 */
modifier whenNotPaused() {
  require(!paused);
  _;
}


/**
 * @dev Modifier to make a function callable only when the contract is paused.
 */
modifier whenPaused() {
  require(paused);
  _;
}


/**
 * @dev called by the owner to pause, triggers stopped state
 */
```

```
  function pause() public onlyOwner whenNotPaused {
    paused = true;
    Pause();
  }

  /**
   * @dev called by the owner to unpause, returns to normal state
   */
  function unpause() public onlyOwner whenPaused {
    paused = false;
    Unpause();
  }
}

/**
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */
library SafeMath {
    /**
     * @dev Multiplies two numbers, throws on overflow.
     */
```

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

    /**
     * @dev Integer division of two numbers, truncating the quotient.
     */
    function div(uint256 a, uint256 b) internal pure returns(uint256) {
        // assert(b > 0); // Solidity automatically throws when dividing by 0
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold
        return c;
    }

    /**
     * @dev Substracts two numbers, throws on overflow (i.e. if subtrahend is greater
than minuend).
```

```
     */
    function sub(uint256 a, uint256 b) internal pure returns(uint256) {
        assert(b <= a);
        return a - b;
    }

    /**
     * @dev Adds two numbers, throws on overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns(uint256) {
        uint256 c = a + b;
        assert(c >= a);
        return c;
    }

}

/*ERC20*/
contract TokenERC20 is Pausable {
    using SafeMath for uint256;
    // Public variables of the token
    string public name = "NRC";
```

```
    string public symbol = "R";
    uint8 public decimals = 0;
    // how many token units a buyer gets per wei
    uint256 public rate = 50000;
    // address where funds are collected
    address public wallet = 0xd3C8326064044c36B73043b009155a59e92477D0;
    // contributors address
    address public contributorsAddress =
0xa7db53CB73DBe640DbD480a928dD06f03E2aE7Bd;
    // company address
    address public companyAddress =
0x9c949b51f2CafC3A5efc427621295489B63D861D;
    // market Address
    address public marketAddress =
0x199EcdFaC25567eb4D21C995B817230050d458d9;
    // share of all token
    uint8 public constant ICO_SHARE = 20;
    uint8 public constant CONTRIBUTORS_SHARE = 30;
    uint8 public constant COMPANY_SHARE = 20;
    uint8 public constant MARKET_SHARE = 30;
    // unfronzen periods
    uint8 constant COMPANY_PERIODS = 10;
```

```
uint8 constant CONTRIBUTORS_PERIODS = 3;

// token totalsupply amount

uint256 public constant TOTAL_SUPPLY = 80000000000;

// ico token amount

uint256 public icoTotalAmount = 16000000000;

uint256 public companyPeriodsElapsed;

uint256 public contributorsPeriodsElapsed;

// token frozened amount

uint256 public frozenSupply;

uint256 public initDate;

uint8 public contributorsCurrentPeriod;

uint8 public companyCurrentPeriod;

// This creates an array with all balances

mapping(address => uint256) public balanceOf;



// This generates a public event on the blockchain that will notify clients

event Transfer(address indexed from, address indexed to, uint256 value);

event InitialToken(string desc, address indexed target, uint256 value);



/**

 * Constrctor function

 * Initializes contract with initial supply tokens to the creator of the contract
```

```
     */
    function TokenERC20(
    ) public {
        // contributors share 30% of totalSupply,but get all by 3 years
        uint256 tempContributors =
TOTAL_SUPPLY.mul(CONTRIBUTORS_SHARE).div(100).div(CONTRIBUTORS_PERIO
DS);
        contributorsPeriodsElapsed = tempContributors;
        balanceOf[contributorsAddress] = tempContributors;
        InitialToken("contributors", contributorsAddress, tempContributors);

        // company shares 20% of totalSupply,but get all by 10 years
        uint256 tempCompany =
TOTAL_SUPPLY.mul(COMPANY_SHARE).div(100).div(COMPANY_PERIODS);
        companyPeriodsElapsed = tempCompany;
        balanceOf[companyAddress] = tempCompany;
        InitialToken("company", companyAddress, tempCompany);

        // ico takes 20% of totalSupply
        uint256 tempIco = TOTAL_SUPPLY.mul(ICO_SHARE).div(100);
        icoTotalAmount = tempIco;
```

```
        // expand the market cost 30% of totalSupply
        uint256 tempMarket = TOTAL_SUPPLY.mul(MARKET_SHARE).div(100);
        balanceOf[marketAddress] = tempMarket;
        InitialToken("market", marketAddress, tempMarket);

        // frozenSupply waitting for being unfrozen
        uint256 tempFrozenSupply = TOTAL_SUPPLY.sub(tempContributors).sub(tempIco).sub(tempCompany).sub(tempMarket);
        frozenSupply = tempFrozenSupply;
        initDate = block.timestamp;
        contributorsCurrentPeriod = 1;
        companyCurrentPeriod = 1;
        paused = true;
    }

    /**
     * Internal transfer, only can be called by this contract
     */
    function _transfer(address _from, address _to, uint _value) internal {
        // Prevent transfer to 0x0 address. Use burn() instead
        require(_to != 0x0);
```

```
        // Check if the sender has enough
        require(balanceOf[_from] >= _value);
        // Check for overflows
        require(balanceOf[_to].add(_value) > balanceOf[_to]);
        // Save this for an assertion in the future
        uint previousBalances = balanceOf[_from].add(balanceOf[_to]);
        // Subtract from the sender
        balanceOf[_from] = balanceOf[_from].sub(_value);
        // Add the same to the recipient
        balanceOf[_to] = balanceOf[_to].add(_value);
        Transfer(_from, _to, _value);
        // Asserts are used to use static analysis to find bugs in your code. They
should never fail
        assert(balanceOf[_from].add(balanceOf[_to]) == previousBalances);
    }

    /**
     * Transfer tokens
     *
     * Send `_value` tokens to `_to` from your account
     *
     * @param _to The address of the recipient
```

```
     * @param _value the amount to send
     */
    function transfer(address _to, uint256 _value) public {
        _transfer(msg.sender, _to, _value);
    }
}
/******************************************/
/*          NRCToken STARTS HERE          */
/******************************************/

contract NRCToken is Owned, TokenERC20 {
    uint256 private etherChangeRate = 10 ** 18;
    uint256 private minutesOneYear = 365*24*60 minutes;
    bool public   tokenSaleActive = true;
    // token have been sold
    uint256 public totalSoldToken;
    // all frozenAccount addresses
    mapping(address => bool) public frozenAccount;

    /* This generates a public log event on the blockchain that will notify clients */
    event LogFrozenAccount(address target, bool frozen);
    event LogUnfrozenTokens(string desc, address indexed targetaddress, uint256
```

```
unfrozenTokensAmount);
    event LogSetTokenPrice(uint256 tokenPrice);
    event TimePassBy(string desc, uint256 times );
    /**
     * event for token purchase logging
     * @param purchaser who paid for the tokens
     * @param value ehter paid for purchase
     * @param amount amount of tokens purchased
     */
    event LogTokenPurchase(address indexed purchaser, uint256 value, uint256
amount);
    // ICO finished Event
    event TokenSaleFinished(string desc, address indexed contributors, uint256
icoTotalAmount, uint256 totalSoldToken, uint256 leftAmount);

    /* Initializes contract with initial supply tokens to the creator of the contract */
    function NRCToken() TokenERC20() public {}

    /* Internal transfer, only can be called by this contract */
    function _transfer(address _from, address _to, uint _value) internal {
        require(_from != _to);
        require(_to != 0x0); // Prevent transfer to 0x0 address. Use burn() instead
```

```
        require(balanceOf[_from] >= _value); // Check if the sender has enough
        require(balanceOf[_to].add(_value) > balanceOf[_to]); // Check for overflows
        require(!frozenAccount[_from]); // Check if sender is frozen
        require(!frozenAccount[_to]); // Check if recipient is frozen
        balanceOf[_from] = balanceOf[_from].sub(_value); // Subtract from the sender
        balanceOf[_to] = balanceOf[_to].add(_value); // Add the same to the recipient
        Transfer(_from, _to, _value);
    }
      /**
     * Transfer tokens
     *
     * Send `_value` tokens to `_to` from your account
     *
     * @param _to The address of the recipient
     * @param _value the amount to send
     */
    function transfer(address _to, uint256 _value) public {
        _transfer(msg.sender, _to, _value);
    }
```

```
    /// @notice `freeze? Prevent | Allow` `target` from sending & receiving tokens
    /// @param target Address to be frozen
    /// @param freeze either to freeze it or not
    function freezeAccount(address target, bool freeze) public onlyOwner whenNotPaused {
        require(target != 0x0);
        require(target != owner);
        require(frozenAccount[target] != freeze);
        frozenAccount[target] = freeze;
        LogFrozenAccount(target, freeze);
    }

    /// @notice Allow users to buy tokens for `newTokenRate` eth
    /// @param newTokenRate Price users can buy from the contract
    function setPrices(uint256 newTokenRate) public onlyOwner whenNotPaused {
        require(newTokenRate > 0);
        require(newTokenRate <= icoTotalAmount);
        require(tokenSaleActive);
        rate = newTokenRate;
        LogSetTokenPrice(newTokenRate);
    }
```

```
    /// @notice Buy tokens from contract by sending ether
    function buy() public payable whenNotPaused {
        // if ICO finished ,can not buy any more!
        require(!frozenAccount[msg.sender]);
        require(tokenSaleActive);
        require(validPurchase());
        uint tokens = getTokenAmount(msg.value); // calculates the amount
        require(!validSoldOut(tokens));
        LogTokenPurchase(msg.sender, msg.value, tokens);
        balanceOf[msg.sender] = balanceOf[msg.sender].add(tokens);
        calcTotalSoldToken(tokens);
        forwardFunds();
    }

    // Override this method to have a way to add business logic to your crowdsale
when buying
    function getTokenAmount(uint256 etherAmount) internal view returns(uint256) {
        uint256 temp = etherAmount.mul(rate);
        uint256 amount = temp.div(etherChangeRate);
        return amount;
    }
```

```
// send ether to the funder wallet
function forwardFunds() internal {
    wallet.transfer(msg.value);
}

// calc totalSoldToken
function calcTotalSoldToken(uint256 soldAmount) internal {
    totalSoldToken = totalSoldToken.add(soldAmount);
    if (totalSoldToken >= icoTotalAmount) {
        tokenSaleActive = false;
    }
}

// @return true if the transaction can buy tokens
function validPurchase() internal view returns(bool) {
    bool limitPurchase = msg.value >= 1 ether;
    bool isNotTheOwner = msg.sender != owner;
    bool isNotTheCompany = msg.sender != companyAddress;
    bool isNotWallet = msg.sender != wallet;
    bool isNotContributors = msg.sender != contributorsAddress;
    bool isNotMarket = msg.sender != marketAddress;
    return limitPurchase && isNotTheOwner && isNotTheCompany &&
```

```
isNotWallet && isNotContributors && isNotMarket;
    }

    // @return true if the ICO is in progress.
    function validSoldOut(uint256 soldAmount) internal view returns(bool) {
        return totalSoldToken.add(soldAmount) > icoTotalAmount;
    }
    // @return current timestamp
    function time() internal constant returns (uint) {
        return block.timestamp;
    }

    /// @dev send the rest of the tokens after the crowdsale end and
    /// send to contributors address
    function finaliseICO() public onlyOwner whenNotPaused {
        require(tokenSaleActive == true);
        uint256 tokensLeft = icoTotalAmount.sub(totalSoldToken);
        tokenSaleActive = false;
        require(tokensLeft > 0);
        balanceOf[contributorsAddress]                                    =
balanceOf[contributorsAddress].add(tokensLeft);
        TokenSaleFinished("finaliseICO",    contributorsAddress,    icoTotalAmount,
```

```
totalSoldToken, tokensLeft);
            totalSoldToken = icoTotalAmount;
        }



        /// @notice freeze unfrozenAmount
        function unfrozenTokens() public onlyOwner whenNotPaused {
            require(frozenSupply >= 0);
            if (contributorsCurrentPeriod < CONTRIBUTORS_PERIODS) {
                unfrozenContributorsTokens();
                unfrozenCompanyTokens();
            } else {
                unfrozenCompanyTokens();
            }
        }


        // unfrozen contributors token year by year
        function unfrozenContributorsTokens() internal {
            require(contributorsCurrentPeriod < CONTRIBUTORS_PERIODS);
            uint256 contributortimeShouldPassBy = contributorsCurrentPeriod *
(minutesOneYear);
            TimePassBy("contributortimeShouldPassBy", contributortimeShouldPassBy);
```

```
        uint256 contributorsTimePassBy = time() - initDate;
        TimePassBy("contributortimePassBy", contributorsTimePassBy);

        contributorsCurrentPeriod = contributorsCurrentPeriod + 1;
        require(contributorsTimePassBy >= contributortimeShouldPassBy);
        frozenSupply = frozenSupply.sub(contributorsPeriodsElapsed);
        balanceOf[contributorsAddress] =
balanceOf[contributorsAddress].add(contributorsPeriodsElapsed);
        LogUnfrozenTokens("contributors", contributorsAddress,
contributorsPeriodsElapsed);
    }

    // unfrozen company token year by year
    function unfrozenCompanyTokens() internal {
        require(companyCurrentPeriod < COMPANY_PERIODS);
        uint256 companytimeShouldPassBy = companyCurrentPeriod *
(minutesOneYear);
        TimePassBy("CompanytimeShouldPassBy", companytimeShouldPassBy);
        uint256 companytimePassBy = time() - initDate;
        TimePassBy("CompanytimePassBy", companytimePassBy);

        require(companytimePassBy >= companytimeShouldPassBy);
```

```
            companyCurrentPeriod = companyCurrentPeriod + 1;
            frozenSupply = frozenSupply.sub(companyPeriodsElapsed);
            balanceOf[companyAddress] =
balanceOf[companyAddress].add(companyPeriodsElapsed);
            LogUnfrozenTokens("company", companyAddress,
companyPeriodsElapsed);
        }

        // fallback function – do not allow any eth transfers to this contract
        function() external {
            revert();
        }

}
```