Beosin

成都链安

BEOSIN

Smart contract security audit report

Audit Number：201901041635

Smart Contract Name：

VESTELLA (VES)

Smart Contract Address：

NULL

Smart Contract Address Link：

NULL

Start Date：2019.01.02

Completion Date：2019.01.04

Overall Result：Pass（Distinction）

Audit Team: Beosin (Chengdu LianAn) Technology Co. Ltd.

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	ERC20 Token Standards	Pass
		Compiler Version Security	Pass
		Visibility Specifiers	Pass
		Gas Consumption	Pass
		SafeMath Features	Pass
		Fallback Usage	Pass
		tx.origin Usage	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		Overriding Variables	Pass
2	Function Call Audit	Authorization of Function Call	Pass
		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass

		selfdestruct Function Security	Pass
3	Business Security	Access Control of Owner	Pass
		Business Logics	Pass
		Business Implementations	Pass
4	Integer Overflows/Underflows	-	Pass
5	Reentrancy	-	Pass
6	Exceptional Reachable State	-	Pass
7	Transaction-Ordering Dependence	-	Pass
8	Block Properties Dependence	-	Pass
9	Pseudo-random Number Generator (PRNG)	-	Pass
10	DoS (Denial of Service)	-	Pass
11	Token Vesting Implementation	-	Pass
12	Fake Deposit	-	Pass
13	event security	-	Pass

Note: Audit results and suggestions in code comments

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of smart contract VES. Including Coding Standards, Security, and Business Logic. **VES contract passed all test items. The overall result is Pass (Distinction). The smart contract is able to function properly.** Please find below the basic information of the smart contract:

1、Basic Token Information

Token name	VESTELLA
Token symbol	VES
decimals	18
totalSupply	15 Billion (Can be destroyed, excessive issuing will not be permitted)
owner	0x0F1b590cD3155571C8680B363867e20b8E4303bE
Token type	ERC20

Table 1 – Basic Token Information

2、Token Vesting Information

There are 3 kinds of way to participate Token Vesting. As shown in Table 2. Project side (VESTELLA) can add lock position for any appointed address of account.

Round	Time	Holder
Pre-Round	3 months	Only for Bonus
Private-Round	12 months	Only for Bonus
Team-Round	24 months	Partners, Advisors

Table 2 – Token Vesting Information

3、Customized Function Audit:

As shown in Figure 1, the internal function '_batchTransfer' can transfer a specific amount of tokens to different accounts. It can be called by airdrop function to complete 'airdrop' operation(As shown in Figure 2). Implementation of the code is secure and has no risks, fits the original design.

```
477     function _batchTransfer(address[] memory _to, uint256[] memory _amount) internal whenNotPaused {
478         require(_to.length == _amount.length);
479         uint256 sum = 0;
480         for(uint i = 0;i < _to.length;i += 1){
481             require(_to[i] != address(0));
482             sum = sum.add(_amount[i]);
483             require(sum <= balances[msg.sender]);
484             balances[_to[i]] = balances[_to[i]].add(_amount[i]);
485             emit Transfer(msg.sender, _to[i], _amount[i]);
486         }
487         balances[msg.sender] = balances[msg.sender].sub(sum);
488     }
```

Figure 1 – Screenshot of _batchTransfer internal function source code

```
495     function airdrop(address[] memory _to, uint256[] memory _amount) public onlyOwner returns(bool){
496         _batchTransfer(_to, _amount);
497         return true;
498     }
```

Figure 2 – Screenshot of airdriop fuction source code

Audited Source Code with Comments:

```
pragma solidity 0.5.1;


library SafeMath {

    uint256 constant internal MAX_UINT = 2 ** 256 - 1; // max uint256

    /**
     * @dev Multiplies two numbers, reverts on overflow.
     */
    function mul(uint256 _a, uint256 _b) internal pure returns(uint256) {
        if (_a == 0) {
            return 0;
        }
        require(MAX_UINT / _a >= _b);
        return _a * _b;
    }

    /**
     * @dev Integer division of two numbers truncating the quotient, reverts on
division by zero.
     */
    function div(uint256 _a, uint256 _b) internal pure returns(uint256) {
        require(_b != 0);
        return _a / _b;
    }

    /**
     * @dev Subtracts two numbers, reverts on overflow (i.e. if subtrahend is
```

```
greater than minuend).
     */
    function sub(uint256 _a, uint256 _b) internal pure returns(uint256) {
        require(_b <= _a);
        return _a - _b;
    }

    /**
     * @dev Adds two numbers, reverts on overflow.
     */
    function add(uint256 _a, uint256 _b) internal pure returns(uint256) {
        require(MAX_UINT - _a >= _b);
        return _a + _b;
    }

}


contract Ownable {
    address public owner;
    // Beosin (Chengdu LianAn) // Declare the event 'OwnershipTransferred'.
    event OwnershipTransferred(
        address indexed previousOwner,
        address indexed newOwner
    );

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(msg.sender == owner);
        _;
    }

    /**
     * @dev Allows the current owner to transfer control of the contract to a
newOwner.
     * @param _newOwner The address to transfer ownership to.
     */
    function transferOwnership(address _newOwner) public onlyOwner {
        _transferOwnership(_newOwner);
    }

    /**
     * @dev Transfers control of the contract to a newOwner.
     * @param _newOwner The address to transfer ownership to.
     */
    function _transferOwnership(address _newOwner) internal {
```

```
        require(_newOwner != address(0)); // Beosin (Chengdu LianAn) // Require that the ownership can not be transfered to a zero address.
        emit OwnershipTransferred(owner, _newOwner); // Beosin (Chengdu LianAn) // Trigger the event 'OwnershipTransferred'.
        owner = _newOwner;
    }
}


contract Pausable is Ownable {
    // Beosin (Chengdu LianAn) // Declare the event 'Pause' and 'Unpause'.
    event Pause();
    event Unpause();

    bool public paused = false; // Beosin (Chengdu LianAn) // Default 'false', will not effect on using this contract.

    /**
     * @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;
        emit Pause();
    }

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

```
    }
}


contract StandardToken {
    using SafeMath for uint256; // Beosin (Chengdu LianAn) // Pull in the SafeMath library for Mathematical operation.

    mapping(address => uint256) internal balances; // Beosin (Chengdu LianAn) // Mapping for storing the token amount of corresponding address.
    mapping(address => mapping(address => uint256)) internal allowed; // Beosin (Chengdu LianAn) // Mapping for storing the approval value which one address was allowed by another address.

    uint256 internal totalSupply_;
    // Beosin (Chengdu LianAn) // Declare the event 'Transfer'.
    event Transfer(
        address indexed from,
        address indexed to,
        uint256 value
    );
    // Beosin (Chengdu LianAn) // Declare the event 'Approval'.
    event Approval(
        address indexed owner,
        address indexed spender,
        uint256 value
    );

    /**
     * @dev Total number of tokens in existence
     */
    function totalSupply() public view returns(uint256) {
        return totalSupply_;
    }

    /**
     * @dev Gets the balance of the specified address.
     * @param _owner The address to query the the balance of.
     * @return An uint256 representing the amount owned by the passed address.
     */
    function balanceOf(address _owner) public view returns(uint256) {
        return balances[_owner];
    }

    /**
     * @dev Function to check the amount of tokens that an owner allowed to a spender.
     * @param _owner address The address which owns the funds.
```

```
     * @param _spender address The address which will spend the funds.
     * @return A uint256 specifying the amount of tokens still available for the spender.
     */
    function allowance(
        address _owner,
        address _spender
    )
    public
    view
    returns(uint256) {
        return allowed[_owner][_spender];
    }

    /**
     * @dev Transfer token for a specified address
     * @param _to The address to transfer to.
     * @param _value The amount to be transferred.
     */
    function transfer(address _to, uint256 _value) public returns(bool) {
        require(_to != address(0)); // Beosin (Chengdu LianAn) // Require that the target address '_to' is not zero.
        require(_value <= balances[msg.sender]); // Beosin (Chengdu LianAn) // Require that the balances of caller are more than transaction value.
        // Beosin (Chengdu LianAn) // Alter the balances of corresponding address and trigger the event 'Transfer' to complete the 'transfer' transaction.
        balances[msg.sender] = balances[msg.sender].sub(_value);
        balances[_to] = balances[_to].add(_value);
        emit Transfer(msg.sender, _to, _value);
        return true;
    }

    /**
     * @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.sender.
     * @param _spender The address which will spend the funds.
     * @param _value The amount of tokens to be spent.
     */
    // Beosin (Chengdu LianAn) // User call this function can lead to re-approve.
    // Beosin (Chengdu LianAn) // Recommend that first reduce the spender's allowance to 0 and set the desired value afterwards.
    // Beosin (Chengdu LianAn) // Or use function 'increaseApproval' and 'decreaseApproval' to alter approval value.
    function approve(address _spender, uint256 _value) public returns(bool) {
        allowed[msg.sender][_spender] = _value; // Beosin (Chengdu LianAn) // Set the approval value as '_value' and trigger the event 'Approval' to complete the 'approve' operation.
        emit Approval(msg.sender, _spender, _value);
```

```
        return true;
    }

    /**
     * @dev Transfer tokens from one address to another
     * @param _from address The address which you want to send tokens from
     * @param _to address The address which you want to transfer to
     * @param _value uint256 the amount of tokens to be transferred
     */
    function transferFrom(
        address _from,
        address _to,
        uint256 _value
    )
    public
    returns(bool) {
        require(_to != address(0)); // Beosin (Chengdu LianAn) // Require that the target address '_to' is not zero.
        require(_value <= balances[_from]); // Beosin (Chengdu LianAn) // Require that the balances of '_from' are more than transaction value.
        require(_value <= allowed[_from][msg.sender]); // Beosin (Chengdu LianAn) // Require that the approval value '_from' allowed to 'msg.sender' is more than transaction value.
        // Beosin (Chengdu LianAn) // Alter the balances and approval value of corresponding address and trigger the event 'Transfer' to complete the 'transferFrom' transaction.
        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 Increase the amount of tokens that an owner allowed to a spender.
     * @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); // Beosin (Chengdu LianAn) // Increase the approval value , alter value is '_addedValue'.
        emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]); //
```

```
Beosin (Chengdu LianAn) // Trigger the event 'Approval' to complete the 'increaseApproval' operation.
        return true;
    }

    /**
     * @dev Decrease the amount of tokens that an owner allowed to a spender.
     * @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]; // Beosin (Chengdu LianAn) // Get the old approval value .
        if (_subtractedValue >= oldValue) {
            allowed[msg.sender][_spender] = 0; // Beosin (Chengdu LianAn) // If the value to decrease '_subtractedValue' is more than old approval value, reduce approval value to 0 directly.
        } else {
            allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue); // Beosin (Chengdu LianAn) // Decrease the approval value , alter value is '_subtractedValue '.
        }
        emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]); // Beosin (Chengdu LianAn) // Trigger the event 'Approval' to complete the 'decreaseApproval' operation.
        return true;
    }
}


contract BurnableToken is StandardToken {
    event Burn(address indexed account, uint256 value);

    /**
     * @dev Burns a specific amount of tokens.
     * @param value The amount of token to be burned.
     */
    function burn(uint256 value) public {
        require(balances[msg.sender] >= value); // Beosin (Chengdu LianAn) // Require that balances of 'msg.sender' are more than burn value.
        // Beosin (Chengdu LianAn) // Alter the total supply and balances of corresponding address, trigger the event 'Burn' and 'Transfer' to complete the 'burn' operation.
        totalSupply_ = totalSupply_.sub(value);
```

```
        balances[msg.sender] = balances[msg.sender].sub(value);
        emit Burn(msg.sender, value);
        emit Transfer(msg.sender, address(0), value);
    }

    /**
     * @dev Burns a specific amount of tokens which belong to appointed address of
account.
     * @param account The address of appointed account.
     * @param value The amount of token to be burned.
     */
    function burnFrom(address account, uint256 value) public {
        require(account != address(0)); // Beosin (Chengdu LianAn) // Require that
appointed address is not zero.
        require(balances[account] >= value); // Beosin (Chengdu LianAn) // Require
that balances of 'account' are more than burn value.
        require(allowed[account][msg.sender] >= value); // Beosin (Chengdu LianAn) //
Require that the approval value 'account' allowed to 'msg.sender' is more than burn value.
        // Beosin (Chengdu LianAn) // Alter the total supply, balances and approval value
of corresponding address, trigger the event 'Burn' and 'Transfer' to complete the 'burnFrom'
operation.
        totalSupply_ = totalSupply_.sub(value);
        balances[account] = balances[account].sub(value);
        allowed[account][msg.sender] = allowed[account][msg.sender].sub(value);
        emit Burn(account, value);
        emit Transfer(account, address(0), value);
    }
}


/**
 * @dev Rewrite the key functions, add the modifier 'whenNotPaused',owner can stop
the transaction.
 */
contract PausableToken is StandardToken, Pausable {
    function transfer(
        address _to,
        uint256 _value
    )
    public
    whenNotPaused
    returns(bool) {
        return super.transfer(_to, _value);
    }

    function transferFrom(
        address _from,
        address _to,
```

```
        uint256 _value
    )
    public
    whenNotPaused
    returns(bool) {
        return super.transferFrom(_from, _to, _value);
    }

    function approve(
        address _spender,
        uint256 _value
    )
    public
    whenNotPaused
    returns(bool) {
        return super.approve(_spender, _value);
    }

    function increaseApproval(
        address _spender,
        uint _addedValue
    )
    public
    whenNotPaused
    returns(bool success) {
        return super.increaseApproval(_spender, _addedValue);
    }

    function decreaseApproval(
        address _spender,
        uint _subtractedValue
    )
    public
    whenNotPaused
    returns(bool success) {
        return super.decreaseApproval(_spender, _subtractedValue);
    }
}


/**
 * @title VESTELLAToken token contract
 * @dev Initialize the basic information of VESTELLAToken.
 */
contract VESTELLAToken is PausableToken, BurnableToken {
    using SafeMath for uint256;

    string public constant name = "VESTELLA"; // name of Token
```

```
    string public constant symbol = "VES"; // symbol of Token
    uint8 public constant decimals = 18; // decimals of Token
    uint256 constant _INIT_TOTALSUPPLY = 15000000000;

    mapping (address => uint256[]) internal locktime; // Beosin (Chengdu LianAn) // Mapping for storing all the lock-time of locked position with corresponding address.
    mapping (address => uint256[]) internal lockamount; // Beosin (Chengdu LianAn) // Mapping for storing all the lock-amount of locked position with corresponding address.

    event AddLockPosition(address indexed account, uint256 amount, uint256 time); // Beosin (Chengdu LianAn) // Declare the event 'AddLockPosition'.


    /**
     * @dev constructor Initialize the basic information.
     */
    constructor() public {
        totalSupply_ = _INIT_TOTALSUPPLY * 10 ** uint256(decimals); // Beosin (Chengdu LianAn) // Total amount 15 billion, excessive issuing will not be permitted.
        owner = 0x0F1b590cD3155571C8680B363867e20b8E4303bE; // Beosin (Chengdu LianAn) // Appoint owner address to manage all the token.
        balances[owner] = totalSupply_;
    }

    /**
     * @dev addLockPosition function that only owner can add lock position for appointed address of account.
     * one address can participate more than one lock position plan.
     * @param account The address of account will participate lock position plan.
     * @param amount The array of token amount that will be locked.
     * @param time The timestamp of token will be released.
     */
    function addLockPosition(address account, uint256[] memory amount, uint256[] memory time) public onlyOwner returns(bool) {
        require(account != address(0)); // Beosin (Chengdu LianAn) // Require that appointed address is not zero.
        require(amount.length == time.length); // Beosin (Chengdu LianAn) // Require that the length of array 'amount' and 'time' is equal.
        uint256 _lockamount = 0;
        for(uint i = 0; i < amount.length; i++) {
            uint256 _amount = amount[i] * 10 ** uint256(decimals);
            require(time[i] > now); // Beosin (Chengdu LianAn) // Require that lock-time is far from now.
            // Beosin (Chengdu LianAn) // Storing the information of locked position to array, and trigger the event 'AddLockPosition' to complete 'addLockPosition' operation.
            locktime[account].push(time[i]);
            lockamount[account].push(_amount);
            emit AddLockPosition(account, _amount, time[i]);
```

```
            _lockamount = _lockamount.add(_amount); // Beosin (Chengdu LianAn) // Get the total lock-amount.
        }
        require(balances[msg.sender] >= _lockamount); // Beosin (Chengdu LianAn) // Require that balances of 'msg.sender' is more than the amount to transfer '_lockamount'.
        // Beosin (Chengdu LianAn) // Alter balances of corresponding address, trigger the event 'Transfer'.
        balances[account] = balances[account].add(_lockamount);
        balances[msg.sender] = balances[msg.sender].sub(_lockamount);
        emit Transfer(msg.sender, account, _lockamount);
        return true;
    }

    /**
     * @dev getLockPosition function get the detail information of an appointed account.
     * @param account The address of appointed account.
     */
    function getLockPosition(address account) public view returns(uint256[] memory _locktime, uint256[] memory _lockamount) {
        return (locktime[account], lockamount[account]);
    }

    /**
     * @dev getLockedAmount function get the amount of locked token which belong to appointed address at the current time.
     * @param account The address of appointed account.
     */
    function getLockedAmount(address account) public view returns(uint256 _lockedAmount) {
        uint256 _Amount = 0;
        uint256 _lockAmount = 0;
        for(uint i = 0; i < locktime[account].length; i++) {
            if(now < locktime[account][i]) {
                _Amount = lockamount[account][i];
                _lockAmount = _lockAmount.add(_Amount);
            }
        }
        return _lockAmount;
    }

    /**
     * @dev Rewrite the transfer functions, call the getLockedAmount to validate the balances after transaction is more than lock-amount.
     */
    function transfer(
        address _to,
        uint256 _value
```

```
    )
    public
    returns(bool) {
        require(balances[msg.sender].sub(_value) >= getLockedAmount(msg.sender));
        return super.transfer(_to, _value);
    }

    /**
     * @dev Rewrite the transferFrom functions, call the getLockedAmount to validate
the balances after transaction is more than lock-amount.
     */
    function transferFrom(
        address _from,
        address _to,
        uint256 _value
    )
    public
    returns(bool) {
        require(balances[_from].sub(_value) >= getLockedAmount(_from));
        return super.transferFrom(_from, _to, _value);
    }

    /**
     * @dev Rewrite the burn functions, call the getLockedAmount to validate the
balances after burning is more than lock-amount.
     */
    function burn(uint256 value) public {
        require(balances[msg.sender].sub(value) >= getLockedAmount(msg.sender));
        super.burn(value);
    }

    /**
     * @dev Rewrite the burnFrom functions, call the getLockedAmount to validate the
balances after burning is more than lock-amount.
     */
    function burnFrom(address account, uint256 value) public {
        require(balances[account].sub(value) >= getLockedAmount(account));
        super.burnFrom(account, value);
    }

    /**
     * @dev _batchTransfer internal function for airdropping candy to target
address.
     * @param _to target address
     * @param _amount amount of token
     */
    function _batchTransfer(address[] memory _to, uint256[] memory _amount) internal
whenNotPaused {
```

```
        require(_to.length == _amount.length); // Beosin (Chengdu LianAn) // Require that the length of array '_to' and '_amount' is equal.
        uint256 sum = 0;
        for(uint i = 0;i < _to.length;i += 1){
            require(_to[i] != address(0)); // Beosin (Chengdu LianAn) // Require that the target address '_to' is not zero.
            sum = sum.add(_amount[i]); // Beosin (Chengdu LianAn) // Get the total amount to transfer.
            require(sum <= balances[msg.sender]); // Beosin (Chengdu LianAn) // Require that the balances of 'msg.sender' is more than total amount to transfer.
            // Beosin (Chengdu LianAn) // Alter the balances of '_to[i]', trigger the event 'Transfer'.
            balances[_to[i]] = balances[_to[i]].add(_amount[i]);
            emit Transfer(msg.sender, _to[i], _amount[i]);
        }
        balances[msg.sender] = balances[msg.sender].sub(sum); // Beosin (Chengdu LianAn) // Alter the balances of 'msg.sender'.
    }

    /**
     * @dev airdrop function for airdropping candy to target address.
     * @param _to target address
     * @param _amount amount of token
     */
    function airdrop(address[] memory _to, uint256[] memory _amount) public onlyOwner returns(bool){
        _batchTransfer(_to, _amount); // Beosin (Chengdu LianAn) // Call the internal function '_batchTransfer' to complete 'airdrop' operation.
        return true;
    }
}
```

成都链安
BEOSIN

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/LianAnTech