Smart Contract Security Audit Report

The SlowMist Security Team received the King DAG team's application for smart contract security audit of the KDAG on April 22, 2020. The following are the details and results of this smart contract security audit:

Token name :

KDAG

The Contract address :

0x95e40e065afb3059dcabe4aaf404c1f92756603a

Link address :

https://etherscan.io/address/0x95e40e065afb3059dcabe4aaf404c1f92756603a

The audit items and results :

(Other unknown security vulnerabilities are not included in the audit responsibility scope)

No.	Audit Items	Audit Subclass	Audit Subclass Result
1	Overflow Audit	-	Passed
2	Race Conditions Audit	-	Passed
3	Authority Control Audit	Permission vulnerability audit	Passed
		Excessive auditing authority	Passed
4	Safety Design Audit	Zeppelin module safe use	Passed
		Compiler version security	Passed
		Hard-coded address security	Passed
		Fallback function safe use	Passed
		Show coding security	Passed
		Function return value security	Passed
		Call function security	Passed
5	Denial of Service Audit	-	Passed
6	Gas Optimization Audit	-	Passed
7	Design Logic Audit	-	Passed
8	“False Deposit” vulnerability Audit	-	Passed

9	Malicious Event Log Audit	-	Passed
10	Scoping and Declarations Audit	-	Passed
11	Replay Attack Audit	ECDSA's Signature Replay Audit	Passed
12	Uninitialized Storage Pointers Audit	-	Passed
13	Arithmetic Accuracy Deviation Audit	-	Passed

Audit Result : **Passed**

Audit Number : 0X002004240001

Audit Date : April 24, 2020

Audit Team : SlowMist Security Team

Summary: This is a token contract that does not contain the tokenVault section. The total amount of tokens in the contract remains unchanged. The owner can add any user to the blacklist. The owner can freeze any user account. It is suggested to set the owner to MultiSig contract to reduce the risk of being attacked. SafeMath security module is used, which is a commendable approach. The contract does not have the Overflow and the Race Conditions issue.

During the audit we found some issues:

1. **The frozen state of [to] iss not checked.**
2. **The frozen states of [msg.sender] and [to] are not checked.**

After feeding back with project side, issues listed above are within tolerable limits.

The source code:

```
/**
 *Submitted for verification at Etherscan.io on 2020-01-17
*/
```

//SlowMist// The contract does not have the Overflow and the Race Conditions issue

```
pragma solidity ^0.5.12;



/**
 * @title ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/20
 */
contract ERC20 {
    function totalSupply() public view returns (uint256);
    function balanceOf(address _who) public view returns (uint256);
    function allowance(address _owner, address _spender) public view returns (uint256);
    function transfer(address _to, uint256 _value) public returns (bool);
    function approve(address _spender, uint256 _value) public returns (bool);
    function transferFrom(address _from, address _to, uint256 _value) public returns (bool);

    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
}



/**
 * @title SafeMath
 * @dev Math operations with safety checks that revert on error
 */
```

//SlowMist// SafeMath security Module is used, which is a recommend approach

```
library SafeMath {
    /**
     * @dev Multiplies two numbers, reverts on overflow.
     */
    function mul(uint256 _a, uint256 _b) internal pure returns (uint256) {
        uint256 c = _a * _b;
        require(_a == 0 || c / _a == _b);

        return c;
    }
```

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

    /**
     * @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);
        uint256 c = _a - _b;

        return c;
    }

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

        return c;
    }
}


/**
  * @title Ownable
  * @dev The Ownable contract has an owner address, and provides basic authorization control
  * functions, this simplifies the implementation of "user permissions".
  */
contract Ownable {
    address public owner;

    event OwnershipTransferred(address indexed previousOwner,address indexed newOwner);

    /**
```

```
    * @dev The Ownable constructor sets the original `owner` of the contract to the sender
    * account.
    */
    constructor() public {
        owner = msg.sender;
    }

    /**
    * @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 {
        require(_newOwner != address(0), "New owner cannot be address(0)"); //SlowMist// This check is quite
good in avoiding losing control of the contract caused by user mistakes
        emit OwnershipTransferred(owner, _newOwner);
        owner = _newOwner;
    }
}


/**
* @title Blacklisted
* @dev allow contract owner or administator to add/remove address to/from the blacklist
*/
contract Blacklisted is Ownable {
    mapping (address => bool) public blacklist;

    event SetBlacklist(address indexed _address, bool _bool);

    /**
     * @dev Modifier: throw if _address is in the blacklist
     */
```

```
    modifier notInBlacklist(address _address) {
        require(!blacklist[_address], "Is in Blacklist");
        _;
    }

    /**
     * @dev call by the owner, set/unset single _address into the blacklist
     */
    function setBlacklist(address _address, bool _bool) public onlyOwner {
        require(_address != address(0));

        if(_bool) {
            require(!blacklist[_address], "Already in blacklist");
        } else {
            require(blacklist[_address], "Not in blacklist yet");
        }

        blacklist[_address] = _bool;
        emit SetBlacklist(_address, _bool);
    }
}


/**
 * @title Standard ERC20 token
 *
 * @dev Implementation of the basic standard token.
 * https://github.com/ethereum/EIPs/issues/20
 */
contract StandardToken is ERC20, Blacklisted {
    using SafeMath for uint256;

    mapping(address => uint256) balances;

    mapping (address => mapping (address => uint256)) internal allowed;

    uint256 totalSupply_;

    /**
    * @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) notInBlacklist(msg.sender) notInBlacklist(_to) public returns (bool) {
        require(_to != address(0)); //SlowMist// This kind of check is very good, avoiding user mistake leading to the loss of token during transfer

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

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

        return true; //SlowMist// The return value conforms to the EIP20 specification
    }

    /**
```

```
    * @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) notInBlacklist(msg.sender) notInBlacklist(_from) notInBlacklist(_to) public returns (bool) {

        require(_to != address(0)); //SlowMist// This kind of check is very good, avoiding user mistake leading to the loss of token during transfer

        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; //SlowMist// The return value conforms to the EIP20 specification

    }

    /**
    * @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.
    */
    function approve(address _spender, uint256 _value) public returns (bool) {
        require(_value == 0 || allowed[msg.sender][_spender] == 0 );
        allowed[msg.sender][_spender] = _value;

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

        return true; //SlowMist// The return value conforms to the EIP20 specification

    }

    /**
    * @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)
    * @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)
    * @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;
    }
}


/**
 * @title Pausable token
 * @dev StandardToken can be frozen.
 **/
contract PausableToken is StandardToken {

    mapping (address => bool) public frozenAccount;
    event FrozenFunds(address target, bool frozen);

    //SlowMist// The frozen state of [_to] is not checked

    function transfer(address _to, uint256 _value) public returns (bool)
```

```
    {
        require(!frozenAccount[msg.sender]);
        return super.transfer(_to, _value);
    }

    function freezeAccount(address target, bool freeze) onlyOwner public {
        frozenAccount[target] = freeze;
        emit FrozenFunds(target, freeze);
    }

    //SlowMist// The frozen states of [msg.sender] and [_to] are not checked
    function transferFrom(address _from, address _to, uint256 _value) public returns (bool)
    {
        require(!frozenAccount[_from]);
        return super.transferFrom(_from, _to, _value);
    }


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

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

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


/**
* @title KingDAGToken
* @dev KingDAGToken main contract
*/
contract KingDAGToken is PausableToken {
    string public constant name = "King DAG";
```

```
    string public constant symbol = "KDAG";
    uint8 public constant decimals = 18;
    uint256 public constant INITIAL_SUPPLY = 1000000000;

    constructor() public {
        totalSupply_ = INITIAL_SUPPLY * (10 ** uint256(decimals));
        balances[msg.sender] = totalSupply_;
        emit Transfer(address(0), msg.sender, totalSupply_);
    }
}
```

慢雾科技
slow mist

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

WeChat Official Account