



Smart Contract Security Audit Report



The SlowMist Security Team received the ProjectWITH team's application for smart contract security audit of the WKEN on January 17, 2020. The following are the details and results of this smart contract security audit:

Token name :

WKEN

File name and HASH(SHA256) :

TokenSmartContract.sol:

f2fdef7bbde3980b090515348c615e1aa54f8371d2cab7e30771426c08925359

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 : 0X002001220001

Audit Date : January 22, 2020

Audit Team : SlowMist Security Team

(**Statement** : SlowMist only issues this report based on the fact that has occurred or existed before the report is issued, and bears the corresponding responsibility in this regard. For the facts occur or exist later after the report, SlowMist cannot judge the security status of its smart contract. SlowMist is not responsible for it. The security audit analysis and other contents of this report are based on the documents and materials provided by the information provider to SlowMist as of the date of this report (referred to as "the provided information"). SlowMist assumes that: there has been no information missing, tampered, deleted, or concealed. If the information provided has been missed, modified, deleted, concealed or reflected and is inconsistent with the actual situation, SlowMist will not bear any responsibility for the resulting loss and adverse effects. SlowMist will not bear any responsibility for the background or other circumstances of the project.)

Summary: This is a token contract that does not contain the tokenVault section. The total amount of contract tokens can be changed but there is a cap. Owner can mint tokens through the mint function. Users can burn their tokens through the burn function. SafeMath security module is used, which is a commendable approach. The contract does not have the Overflow and the Race Conditions issue.

The source code:

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

```
pragma solidity >=0.5.0 <0.6.0;
```

```
/**
```

```
 * @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 OwnershipRenounced(address indexed previousOwner);
    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)); //SlowMist// This check is quite good in avoiding losing
    }

    control of the contract caused by user mistakes

    emit OwnershipTransferred(owner, newOwner);
    owner = newOwner;
}

/**
 * @dev Allows the current owner to relinquish control of the contract.
 */
function renounceOwnership() public onlyOwner {
    emit OwnershipRenounced(owner);
}
```

```
        owner = address(0);
    }
}

/**
 * @title Pausable
 * @dev Base contract which allows children to implement an emergency stop mechanism.
 */
contract Pausable is Ownable {
    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 makeWhitelist 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() onlyOwner whenNotPaused public {
        paused = true;
        emit Pause();
    }

    /**
```

```

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

```

```

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

```

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

```

library SafeMath {
    /**
     * @dev Multiplies two numbers, throws on overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256 c) {
        if (a == 0) {
            return 0;
        }
        c = a * b;

        assert(c / a == b); //SlowMist// It is recommended to replace "assert" with "require" to

```

optimize Gas

```

        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 a / b;
    }

    /**

```

```
* @dev Subtracts 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); //SlowMist// It is recommended to replace "assert" with "require" to
```

optimize Gas

```
    return a - b;
```

```
}
```

```
/**
```

```
* @dev Adds two numbers, throws on overflow.
```

```
*/
```

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

```
    c = a + b;
```

```
    assert(c >= a); //SlowMist// It is recommended to replace "assert" with "require" to
```

optimize Gas

```
    return c;
```

```
}
```

```
}
```

```
/**
```

```
* @title ERC20Basic
```

```
* @dev Simpler version of ERC20 interface
```

```
* @dev see https://github.com/ethereum/EIPs/issues/179
```

```
*/
```

```
contract ERC20Basic {
```

```
    function totalSupply() public view returns (uint256);
```

```
    function balanceOf(address who) public view returns (uint256);
```

```
    function transfer(address to, uint256 value) public returns (bool);
```

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

```
}
```

```
/**
```

```
* @title Basic tokens
```

```
* @dev Basic version of StandardToken, with no allowances.
```

```
*/
contract BasicToken is ERC20Basic {
    using SafeMath for uint256;

    mapping(address => uint256) balances;

    uint256 totalSupply_;

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

    /**
     * @dev transfer tokens 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)); //SlowMist// This kind of check is very good, avoiding user mistake

leading to the loss of token during transfer

        require(_value <= balances[msg.sender]);

        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 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];
    }
}
```

```
}

/**
 * @title ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/20
 */
contract ERC20 is ERC20Basic {
    function allowance(address owner, address spender) public view returns (uint256);
    function transferFrom(address from, address to, uint256 value) public returns (bool);
    function approve(address spender, uint256 value) public returns (bool);
    event Approval(address indexed owner, address indexed spender, uint256 value);
}

/**
 * @title Standard ERC20 tokens
 *
 * @dev Implementation of the basic standard tokens.
 * @dev https://github.com/ethereum/EIPs/issues/20
 * @dev Based on code by FirstBlood:
 * https://github.com/Firstbloodio/token/blob/master/smart_contract/FirstBloodToken.sol
 */
contract StandardToken is ERC20, BasicToken {
    mapping (address => mapping (address => uint256)) internal allowed;

    /**
     * @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)); //SlowMist// This kind of check is very good, avoiding user mistake
leading to the loss of token during transfer

        require(_value <= balances[_from]);
        require(_value <= allowed[_from][msg.sender]);

        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.
```

```
 *
```

```
 * Beware that changing an allowance with this method brings the risk that someone may use both the old
```

```
 * and the new allowance by unfortunate transaction ordering. One possible solution to mitigate this
```

```
 * race condition is to first reduce the spender's allowance to 0 and set the desired value afterwards:
```

```
 * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
```

```
 * @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) {
```

```
    allowed[msg.sender][_spender] = _value;
```

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

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

```
}
```

```
/**
```

```
 * @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 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, uint _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, uint _subtractedValue) public returns (bool) {
    uint 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 Mintable token
* @dev Simple ERC20 Token example, with mintable token creation
* @dev Issue: * https://github.com/OpenZeppelin/openzeppelin-solidity/issues/120
* Based on code by TokenMarketNet: https://github.com/TokenMarketNet/ico/blob/master/contracts/MintableToken.sol
*/
contract MintableToken is StandardToken, Ownable {
    event Mint(address indexed to, uint256 amount);
    event MintFinished();
}
```

```
bool public mintingFinished = false;

modifier canMint() {
    require(!mintingFinished);
    _;
}

/**
 * @dev Function to mint tokens
 * @param _to The address that will receive the minted tokens.
 * @param _amount The amount of tokens to mint.
 * @return A boolean that indicates if the operation was successful.
 */
function mint(address _to, uint256 _amount) onlyOwner canMint public returns (bool) {
    require(totalSupply_+_amount <= (1200000000 * (10 ** uint256(18))), "TotalSupply should not be more than INITIAL_SUPPLY");
    totalSupply_ = totalSupply_+_amount;
    balances[_to] = balances[_to].add(_amount);
    emit Mint(_to, _amount);
    emit Transfer(address(0), _to, _amount);
    return true;
}

/**
 * @dev Function to stop minting new tokens.
 * @return True if the operation was successful.
 */
function finishMinting() onlyOwner canMint public returns (bool) {
    mintingFinished = true;
    emit MintFinished();
    return true;
}
}

/**
 * @title Burnable Token
 * @dev Token that can be irreversibly burned (destroyed).
 */
contract BurnableToken is BasicToken, Pausable {

    event Burn(address indexed burner, uint256 value);
```

```
/**
 * @dev Burns a specific amount of tokens.
 * @param _value The amount of tokens to be burned.
 */
function burn(uint256 _value) whenNotPaused public {
    _burn(msg.sender, _value);
}

function _burn(address _who, uint256 _value) internal {
    require(_value <= balances[_who]);
    // no need to require value <= totalSupply, since that would imply the
    // sender's balance is greater than the totalSupply, which *should* be an assertion failure

    balances[_who] = balances[_who].sub(_value);
    totalSupply_ = totalSupply_.sub(_value);
    emit Burn(_who, _value);
    emit Transfer(_who, address(0), _value);
}
}

contract Consts {
    string constant TOKEN_NAME = "ProjectWITH";
    string constant TOKEN_SYMBOL = "WKEN";
    uint8 constant TOKEN_DECIMALS = 18;
    uint256 constant TOKEN_AMOUNT = 1200000000;
}

/**
 * @title MainToken
 */
contract MainToken is Consts

    , MintableToken
    , BurnableToken
    {
    string public constant name = TOKEN_NAME; // solium-disable-line uppercase
    string public constant symbol = TOKEN_SYMBOL; // solium-disable-line uppercase
```

```
uint8 public constant decimals = TOKEN_DECIMALS; // solium-disable-line uppercase

uint256 public constant INITIAL_SUPPLY = TOKEN_AMOUNT * (10 ** uint256(decimals));

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



Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

[@SlowMist_Team](https://twitter.com/SlowMist_Team)

WeChat Official Account

