Smart Contract Security Audit Report

[2021]

The SlowMist Security Team received the TURK team's application for smart contract security audit of the Turtle King on 2021.06.18. The following are the details and results of this smart contract security audit:

Token Name :

Turtle King

The contract address :

https://scope.klaytn.com/account/0x8c783809332be7734fa782eb5139861721f77b33?tabId=txList

The audit items and results :

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

NO.	Audit Items	Result
1	Replay Vulnerability	Passed
2	Denial of Service Vulnerability	Passed
3	Race Conditions Vulnerability	Passed
4	Authority Control Vulnerability	Some Risks
5	Integer Overflow and Underflow Vulnerability	Passed
6	Gas Optimization Audit	Passed
7	Design Logic Audit	Passed
8	Uninitialized Storage Pointers Vulnerability	Passed
9	Arithmetic Accuracy Deviation Vulnerability	Passed
10	"False top-up" Vulnerability	Passed
11	Malicious Event Log Audit	Passed
12	Scoping and Declarations Audit	Passed

NO.	Audit Items	Result
13	Safety Design Audit	Passed

Audit Result : Medium Risk

Audit Number : 0x002106220001

Audit Date : 2021.06.18 - 2021.06.22

Audit Team : SlowMist Security Team

Summary conclusion : This is a token contract that does not contain the tokenVault section. The total amount of contract tokens can be changed. SafeMath security module is used, which is a recommended approach. The contract does not have the Overflow and the Race Conditions issue.

During the audit, we found the following issues:

1. The minter role can mint tokens arbitrarily through the mint function. After communicating with the project team, the project team expressed the role of the owner will transfer to a dedicated minter account as community governance. But there is still a certain risk in this approach.
2. The minter can burn the account balance arbitrarily of any user through the burn function. After communicating with the project team, the project team expressed time lock function will be added as community governance.

The source code:

```
// SPDX-License-Identifier: MIT
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
pragma solidity 0.5.6;

/**
 * @dev Interface of the KIP-13 standard, as defined in the
 * [KIP-13](http://kips.klaytn.com/KIPs/kip-13-interface_query_standard).
 *
 * Implementers can declare support of contract interfaces, which can then be
 * queried by others.
 *
```

```
 * For an implementation, see `KIP13`.
 */
interface IKIP13 {
    /**
     * @dev Returns true if this contract implements the interface defined by
     * `interfaceId`. See the corresponding
     * [KIP-13 section](http://kips.klaytn.com/KIPs/kip-13-
interface_query_standard#how-interface-identifiers-are-defined)
     * to learn more about how these ids are created.
     *
     * This function call must use less than 30 000 gas.
     */
    function supportsInterface(bytes4 interfaceId) external view returns (bool);
}

/**
 * @dev Interface of the KIP7 standard as defined in the KIP. Does not include
 * the optional functions; to access them see `KIP7Metadata`.
 * See http://kips.klaytn.com/KIPs/kip-7-fungible_token
 */
contract IKIP7 is IKIP13{
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() public view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) public view returns (uint256);

    /**
     * @dev Returns the decimals of token.
     */
    function decimals() public view returns (uint8);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a `Transfer` event.
     */
    function transfer(address recipient, uint256 amount) public returns (bool);
```

```
    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through `transferFrom`. This is
     * zero by default.
     *
     * This value changes when `approve` or `transferFrom` are called.
     */
    function allowance(address owner, address spender) public view returns (uint256);

    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * > 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
     *
     * Emits an `Approval` event.
     */
    function approve(address spender, uint256 amount) public returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a `Transfer` event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) public
returns (bool);

    /**
    * @dev Moves `amount` tokens from the caller's account to `recipient`.
    */
    function safeTransfer(address recipient, uint256 amount, bytes memory data)
public;

    /**
    * @dev  Moves `amount` tokens from the caller's account to `recipient`.
```

```
    */
    function safeTransfer(address recipient, uint256 amount) public;

    /**
    * @dev Moves `amount` tokens from `sender` to `recipient` using the allowance
mechanism.
    * `amount` is then deducted from the caller's allowance.
    */
    function safeTransferFrom(address sender, address recipient, uint256 amount,
bytes memory data) public;

    /**
    * @dev Moves `amount` tokens from `sender` to `recipient` using the allowance
mechanism.
    * `amount` is then deducted from the caller's allowance.
    */
    function safeTransferFrom(address sender, address recipient, uint256 amount)
public;

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to `approve`. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender, uint256 value);

    function mint(address account, uint256 amount) external;

    function burn(address account, uint256 amount) external;

    function setOwner(address owner) external;
    function setMinter(address minter) external;
}
/**
 * @dev Implementation of the `IKIP13` interface.
 *
 * Contracts may inherit from this and call `_registerInterface` to declare
 * their support of an interface.
```

```
 */
contract KIP13 is IKIP13 {
    /*
     * bytes4(keccak256('supportsInterface(bytes4)')) == 0x01ffc9a7
     */
    bytes4 private constant _INTERFACE_ID_KIP13 = 0x01ffc9a7;

    /**
     * @dev Mapping of interface ids to whether or not it's supported.
     */
    mapping(bytes4 => bool) private _supportedInterfaces;

    constructor () internal {
        // Derived contracts need only register support for their own interfaces,
        // we register support for KIP13 itself here
        _registerInterface(_INTERFACE_ID_KIP13);
    }

    /**
     * @dev See `IKIP13.supportsInterface`.
     *
     * Time complexity O(1), guaranteed to always use less than 30 000 gas.
     */
    function supportsInterface(bytes4 interfaceId) external view returns (bool) {
        return _supportedInterfaces[interfaceId];
    }

    /**
     * @dev Registers the contract as an implementer of the interface defined by
     * `interfaceId`. Support of the actual KIP13 interface is automatic and
     * registering its interface id is not required.
     *
     * See `IKIP13.supportsInterface`.
     *
     * Requirements:
     *
     * - `interfaceId` cannot be the KIP13 invalid interface (`0xffffffff`).
     */
    function _registerInterface(bytes4 interfaceId) internal {
        require(interfaceId != 0xffffffff, "KIP13: invalid interface id");
        _supportedInterfaces[interfaceId] = true;
    }
}
```

```
contract IKIP7Receiver {
    function onKIP7Received(address _operator, address _from, uint256 _amount, bytes
memory _data) public returns (bytes4);
}

contract KlaytnToken is KIP13, IKIP7 {
    event SetOwner(address owner);
    event SetMinter(address minter);
    event Transfer(address indexed from, address indexed to, uint amount);
    event Approval(address indexed holder, address indexed spender, uint amount);

    string private _name = "Orbit Bridge Klaytn Token";
    string private _symbol = "OBKT";
    uint8 private _decimals = 18;
    uint private _totalSupply = 0;

    // ------ KIP7 INTERFACE ------
    bytes4 private constant _KIP7_RECEIVED = 0x9d188c22;
    bytes4 private constant _INTERFACE_ID_KIP7 = 0x65787371;

    mapping (address => uint256) private _balances;
    mapping (address => mapping (address => uint256)) private _allowances;   //
(holder, spender)

    // --- Owner ---
    address public _owner;

    // --- Contracts & Constructor ---
    address public _minter;

    modifier onlyOwner {
        require(msg.sender == _owner);
        _;
    }

    modifier onlyMinter {
        require(msg.sender == _minter || msg.sender == _owner);
        _;
    }

    constructor(string memory name, string memory symbol) public {
        _name = name;
        _symbol = symbol;
        _owner = msg.sender;
        _minter = msg.sender;
```

```
        _registerInterface(_INTERFACE_ID_KIP7);
    }

    function name() public view returns (string memory) {
        return _name;
    }

    function symbol() public view returns (string memory) {
        return _symbol;
    }

    function decimals() public view returns (uint8) {
        return _decimals;
    }

    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }

    function allowance(address owner, address spender) public view returns (uint256)
{
        return _allowances[owner][spender];
    }

    function setOwner(address owner) public onlyOwner {
        require(owner != address(0));
        _owner = owner;
        emit SetOwner(owner);
    }

    function setMinter(address minter) public onlyOwner {
        require(minter != address(0));
        _minter = minter;
        emit SetMinter(minter);
    }

    // --- Math ---
    //SlowMist// SafeMath security module is used, which is a recommend approach
    function safeAdd(uint a, uint b) private pure returns (uint) {
        require(a <= uint(-1) - b);
        return a + b;
```

```
    }

    function safeSub(uint a, uint b) private pure returns (uint) {
        require(a >= b);
        return a - b;
    }

    function transfer(address recipient, uint256 amount) public returns (bool) {
        _transfer(msg.sender, recipient, amount);
        return true;
    }

    function safeTransfer(address recipient, uint256 amount) public {
        safeTransfer(recipient, amount, "");
    }

    function safeTransfer(address recipient, uint256 amount, bytes memory data)
public {
        transfer(recipient, amount);
        require(_checkOnKIP7Received(msg.sender, recipient, amount, data), "KIP7:
transfer to non KIP7Receiver implementer");
    }

    function transferFrom(address sender, address recipient, uint256 amount) public
returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, msg.sender, safeSub(_allowances[sender][msg.sender],
amount));
        //SlowMist// The return value conforms to the KIP7 specification
        return true;
    }

    function safeTransferFrom(address sender, address recipient, uint256 amount)
public {
        safeTransferFrom(sender, recipient, amount, "");
    }

    function safeTransferFrom(address sender, address recipient, uint256 amount,
bytes memory data) public {
        transferFrom(sender, recipient, amount);
        require(_checkOnKIP7Received(sender, recipient, amount, data), "KIP7:
transfer to non KIP7Receiver implementer");
    }

    function approve(address spender, uint256 value) public returns (bool) {
```

```
        _approve(msg.sender, spender, value);
        //SlowMist// The return value conforms to the KIP7 specification
        return true;
    }

    function increaseApproval(address spender, uint256 value) public returns (bool) {
        _approve(msg.sender, spender, safeAdd(_allowances[msg.sender][spender],
value));
        return true;
    }

    function decreaseApproval(address spender, uint256 value) public returns (bool) {
        if(value > _allowances[msg.sender][spender]){
            value = 0;
        }
        else{
            value = safeSub(_allowances[msg.sender][spender], value);
        }

        _approve(msg.sender, spender, value);
        return true;
    }

    function _approve(address owner, address spender, uint256 value) internal {
        require(owner != address(0), "KIP7: approve from the zero address");
        //SlowMist// This kind of check is very good, avoiding user mistake leading
to approve errors
        require(spender != address(0), "KIP7: approve to the zero address");

        _allowances[owner][spender] = value;
        emit Approval(owner, spender, value);
    }

    function _transfer(address sender, address recipient, uint256 amount) internal {
        require(sender != address(0), "KIP7: transfer from the zero address");
        //SlowMist// This kind of check is very good, avoiding user mistake leading
to the loss of token during transfer
        require(recipient != address(0), "KIP7: transfer to the zero address");

        _balances[sender] = safeSub(_balances[sender], amount);
        _balances[recipient] = safeAdd(_balances[recipient], amount);
        emit Transfer(sender, recipient, amount);
    }
    //SlowMist// The owner and minter role can mint tokens arbitrarily through the
mint function
```

```
    function mint(address account, uint256 amount) external onlyMinter{
        _mint(account, amount);
    }
    //SlowMist// The owner and minter role can burn the account balance arbitrarily
of any user through the burn function
    function burn(address account, uint256 amount) external onlyMinter{
        _burn(account, amount);
    }

    function _mint(address account, uint256 amount) internal {
        require(account != address(0), "KIP7: mint to the zero address");

        _totalSupply = safeAdd(_totalSupply, amount);
        _balances[account] = safeAdd(_balances[account], amount);
        emit Transfer(address(0), account, amount);
    }

    function _burn(address account, uint256 value) internal {
        require(account != address(0), "KIP7: burn from the zero address");

        _totalSupply = safeSub(_totalSupply, value);
        _balances[account] = safeSub(_balances[account], value);
        emit Transfer(account, address(0), value);
    }

    function isContract(address account) internal view returns (bool) {
        uint256 size;

        assembly { size := extcodesize(account) }
        return size > 0;
    }

    function _checkOnKIP7Received(address sender, address recipient, uint256 amount,
bytes memory _data)
    internal returns (bool)
    {
        if (!isContract(recipient)) {
            return true;
        }

        bytes4 retval = IKIP7Receiver(recipient).onKIP7Received(msg.sender, sender,
amount, _data);
        return (retval == _KIP7_RECEIVED);
    }
}
```

```
contract TurtleToken is KlaytnToken {
    constructor() public KlaytnToken("Turtle King", "TURK") {
    }
}
```

Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist