Smart Contract Security Audit Report

The SlowMist Security Team received the Humanscape team's application for smart contract security audit of the HUM Token on October 15 , 2019. The following are the details and results of this smart contract security audit:

Token name :

HUM

The Contract address :

0x174afe7a032b5a33a3270a9f6c30746e25708532

Link address :

https://etherscan.io/address/0x174afe7a032b5a33a3270a9f6c30746e25708532

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	-	**Not 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 : **Not Passed**

Audit Number : 0X001910210001

Audit Date : October 21, 2019

Audit Team : SlowMist Security Team

Summary: This is a token contract that does not contain the tokenVault section. SafeMath security module is used, which is a commendable approach. The total amount of the token can be changed, user can burn their token via burn function and the owner of the contract can mint tokens via mint function. During the audit, we found the following problems:

1. The mint function does not set an upper limit, and the project side can mint arbitrarily.

2. The blacklisted modifier does not use in approve function, and the transferFrom function does not restrict to _from and _to address via the modifier, any not-blacklisted address controlled by the blacklisted address can transfer the fund from the blacklisted address after approve

The comprehensive evaluation contract is Not Passed.

The source code:

ERCBasic.sol:

```
pragma solidity ^0.4.23;
```

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

SafeMath.sol:

```
pragma solidity ^0.4.23;



/**
  * @title SafeMath
  * @dev Math operations with safety checks that throw on error
  */
 //SlowMist// SafeMath security module is used, which is a commendable approach
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); //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) {
    uint256 c = a + b;
    assert(c >= a); //SlowMist// It is recommended to replace "assert" with "require" to optimize Gas

    return c;
  }
}
```

Basic Token:

```
pragma solidity ^0.4.23;

import "./ERC20Basic.sol";
import "./SafeMath.sol";
```

```
/**
 * @title Basic token
 * @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 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)); //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 balance) {
        return balances[_owner];
    }

}
```

ERC20.sol:

```
pragma solidity ^0.4.23;

import "./ERC20Basic.sol";


/**
  * @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);
}
```

StanderToken.sol:

```
pragma solidity ^0.4.23;

import "./BasicToken.sol";
import "./ERC20.sol";


/**
  * @title Standard ERC20 token
  *
  * @dev Implementation of the basic standard token.
  * @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.
   */

  //SlowMist// The blacklisted modifier does not use here and the blacklisted address can
approve his fund to others
```

```
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;
  }

}
```

MultiOwnerable.sol:

```
pragma solidity ^0.4.23;

/**
 * @title MultiOwnable
 */
contract MultiOwnable {
  address public root;
  mapping (address => address) public owners; // owner => parent of owner

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

  /**
  * @dev Throws if called by any account other than the owner.
  */
  modifier onlyOwner() {
    require(owners[msg.sender] != 0);
    _;
  }
```

```
  /**
  * @dev Adding new owners
  */
  function newOwner(address _owner) onlyOwner external returns (bool) {
    require(_owner != 0); //SlowMist// This check is quite good in avoiding losing control of the
contract caused by user mistakes
    require(owners[_owner] == 0);
    owners[_owner] = msg.sender;
    return true;
  }

  /**
    * @dev Deleting owners
    */
  function deleteOwner(address _owner) onlyOwner external returns (bool) {
    require(owners[_owner] == msg.sender || (owners[_owner] != 0 && msg.sender == root));
    owners[_owner] = 0;
    return true;
  }
}
```

MintableToken.sol:

```
pragma solidity ^0.4.23;

import "./StandardToken.sol";
import "./MultiOwnable.sol";



/**
  * @title Mintable token
  * @dev Simple ERC20 Token example, with mintable token creation
  * @dev Issue: * https://github.com/OpenZeppelin/zeppelin-solidity/issues/120
  * Based on code by TokenMarketNet:
https://github.com/TokenMarketNet/ico/blob/master/contracts/MintableToken.sol
  */
contract MintableToken is StandardToken, MultiOwnable {
  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.
   */
```

//SlowMist// The mint function does not set an upper limit, and the project side can mint arbitrarily

```
  function mint(address _to, uint256 _amount) onlyOwner canMint public returns (bool) {
```

//SlowMist// require(_to != address(0));

//SlowMist// It's recommended to add the code above, avoiding owner mistake leading to the loss of token during mint

```
    totalSupply_ = totalSupply_.add(_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;
  }
}
```

BurnableToken.sol:

```
pragma solidity ^0.4.23;

import "./BasicToken.sol";


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

  event Burn(address indexed burner, uint256 value);

  /**
   * @dev Burns a specific amount of tokens.
   * @param _value The amount of token to be burned.
   */
  function burn(uint256 _value) 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);
  }
}
```

Blacklisted.sol:

```
pragma solidity ^0.4.23;

import "./MultiOwnable.sol";

/**
```

```
 * @title Basic token
 * @dev Basic version of StandardToken, with no allowances.
 */
contract Blacklisted is MultiOwnable {

  mapping(address => bool) public blacklist;

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

  /**
    * @dev Adds single address to blacklist.
    * @param _villain Address to be added to the blacklist
    */
  function addToBlacklist(address _villain) external onlyOwner {
    blacklist[_villain] = true;
  }

  /**
    * @dev Adds list of addresses to blacklist. Not overloaded due to limitations with truffle testing.
    * @param _villains Addresses to be added to the blacklist
    */
  function addManyToBlacklist(address[] _villains) external onlyOwner {
    for (uint256 i = 0; i < _villains.length; i++) {
      blacklist[_villains[i]] = true;
    }
  }

  /**
    * @dev Removes single address from blacklist.
    * @param _villain Address to be removed to the blacklist
    */
  function removeFromBlacklist(address _villain) external onlyOwner {
    blacklist[_villain] = false;
  }
}
```

HUMToken.sol:

```
pragma solidity ^0.4.23;

import "./MintableToken.sol";
import "./BurnableToken.sol";
import "./Blacklisted.sol";



/**
 * @title HUMToken
 * @dev ERC20 HUMToken.
 * Note they can later distribute these tokens as they wish using `transfer` and other
 * `StandardToken` functions.
 */
contract HUMToken is MintableToken, BurnableToken, Blacklisted {

  string public constant name = "Humanscape"; // solium-disable-line uppercase
  string public constant symbol = "HUM"; // solium-disable-line uppercase
  uint8 public constant decimals = 18; // solium-disable-line uppercase, // 18 decimals is the strongly suggested default, avoid changing it

  uint256 public constant INITIAL_SUPPLY = 1250 * 1000 * 1000 * (10 ** uint256(decimals)); // 1,250,000,000 HUM

  bool public isUnlocked = false;

  /**
   * @dev Constructor that gives msg.sender all of existing tokens.
   */
  constructor(address _wallet) public {

    //SlowMist// require(_wallet != address(0));

    //SlowMist// It's recommended to add the code above, avoiding user mistake leading to the loss of token in constructor

    totalSupply_ = INITIAL_SUPPLY;
    balances[_wallet] = INITIAL_SUPPLY;
    emit Transfer(address(0), _wallet, INITIAL_SUPPLY);
  }

  modifier onlyTransferable() {
    require(isUnlocked || owners[msg.sender] != 0);
```

```
    _;
  }
```

//SlowMist// The transferFrom function does not restrict to _from and _to address,any not-blacklisted address controlled by the blacklisted address can transfer the fund from the blacklisted address after approve

```
  function transferFrom(address _from, address _to, uint256 _value) public onlyTransferable notBlacklisted returns (bool) {
        return super.transferFrom(_from, _to, _value);
  }

  function transfer(address _to, uint256 _value) public onlyTransferable notBlacklisted returns (bool) {
        return super.transfer(_to, _value);
  }

  function unlockTransfer() public onlyOwner {
        isUnlocked = true;
  }

  function lockTransfer() public onlyOwner {
        isUnlocked = false;
  }

}
```

慢雾科技
slow mist

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

WeChat Official Account