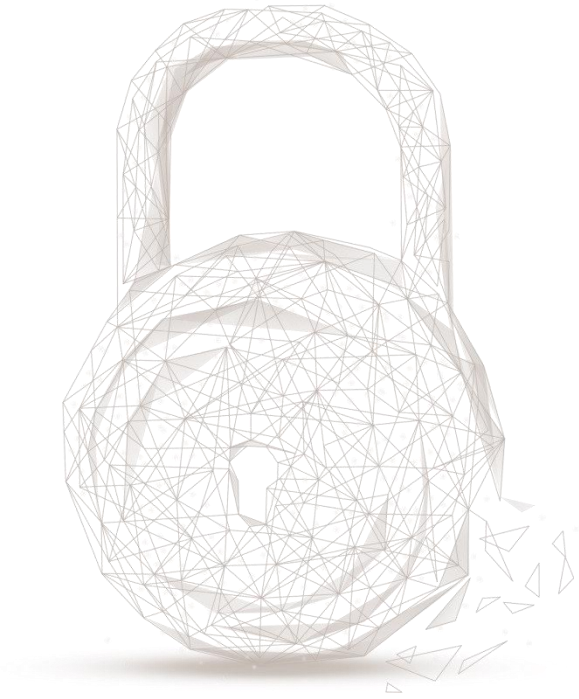
成都链安
BEOSIN

Smart contract security audit report



Audit Number：202009101602

Smart Contract Name：

MoonPool.finance (MFI)

Smart Contract Address：

TLuekEszx5fq61rPCLwga9c3RtAVyBc46j

Smart Contract Address Link：

https://tronscan.org/#/contract/TLuekEszx5fq61rPCLwga9c3RtAVyBc46j/code

Start Date：2020.09.09

Completion Date：2020.09.10

Overall Result：Pass（Distinction）

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

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	TRC20 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 Overflow/Underflow	-	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	-	Missing
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 contracts MFI, including Coding Standards, Security, and Business Logic. **MFI contracts passed all audit 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	MoonPool.finance
Token symbol	MFI
decimals	18
totalSupply	token supply is about 7280 (Mintable with the cap of 10,000)
Token type	TRC20

Table 1 - Basic Token Information

2、Token Vesting Information

Missing

3、Custom Function Audit

- mint

Minting tokens is allowed in this contract, and the address who has the minting permission can mint tokens to the specified address. The minting cap is 10,000, and the total token supply is always less than 10,000.

Audited Source Code with Comments:

```
pragma solidity ^0.5.4; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

interface IERC20 {
    // Beosin (Chengdu LianAn) // Define the function interfaces required by TRC20 Token standard.
    function totalSupply() external view returns (uint);
    function balanceOf(address account) external view returns (uint);
    function transfer(address recipient, uint amount) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint);
    function approve(address spender, uint amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint amount) external returns (bool);
    // Beosin (Chengdu LianAn) // Declare the events 'Transfer' and 'Approval'.
    event Transfer(address indexed from, address indexed to, uint value);
    event Approval(address indexed owner, address indexed spender, uint value);
}

contract Context {
    constructor () internal { }
    // solhint-disable-previous-line no-empty-blocks
    // Beosin (Chengdu LianAn) // Internal function '_msgSender' for getting the caller address.
    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }
}
```

```
contract ERC20 is Context, IERC20 {
    using SafeMath for uint; // Beosin (Chengdu LianAn) // Use the SafeMath library for mathematical operation. Avoid integer overflow/underflow.

    mapping (address => uint) private _balances; // Beosin (Chengdu LianAn) // Declare the mapping variable '_balances' for storing the token balance of corresponding address.

    mapping (address => mapping (address => uint)) private _allowances; // Beosin (Chengdu LianAn) // Declare the mapping variable '_allowances' for storing the allowance between two addresses.

    uint private _totalSupply; // Beosin (Chengdu LianAn) // Declare the variable '_totalSupply' for storing the total token supply.
    uint public maxSupply = 10000 * 1e18; // Beosin (Chengdu LianAn) // Declare the variable 'maxSupply' for storing the cap of minting. It is 10000 as default.
    // Beosin (Chengdu LianAn) // The function 'totalSupply' is defined to query the total token supply.
    function totalSupply() public view returns (uint) {
        return _totalSupply;
    }
    // Beosin (Chengdu LianAn) // The function 'balanceOf' is defined to query the token balance of 'account'.
    function balanceOf(address account) public view returns (uint) {
        return _balances[account];
    }
    // Beosin (Chengdu LianAn) // The 'transfer' function, 'msg.sender' transfers the specified amount of tokens to a specified address.
    function transfer(address recipient, uint amount) public returns (bool) {
        _transfer(_msgSender(), recipient, amount); // Beosin (Chengdu LianAn) // Call the internal function '_transfer' to transfer tokens.
        return true;
    }
    // Beosin (Chengdu LianAn) // The function 'allowance' is defined to query the allowance between 'owner' and 'spender'.
    function allowance(address owner, address spender) public view returns (uint) {
        return _allowances[owner][spender];
    }
    // Beosin (Chengdu LianAn) // The 'approve' function, 'msg.sender' allows the specified amount of tokens to a specified address.
    // Beosin (Chengdu LianAn) // Using functions 'increaseAllowance' and 'decreaseAllowance' to alter allowance is recommended.
    function approve(address spender, uint amount) public returns (bool) {
        _approve(_msgSender(), spender, amount); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to set allowance.
        return true;
    }
    // Beosin (Chengdu LianAn) // The 'transferFrom' function, 'msg.sender' as a delegate of '_from' to transfer the specified amount of tokens to a specified address.
    function transferFrom(address sender, address recipient, uint amount) public returns (bool) {
        _transfer(sender, recipient, amount); // Beosin (Chengdu LianAn) // Call the internal function '_transfer'
```

```
to transfer tokens.
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance")); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to alter the allowance between two addresses.
        return true;
    }
    // Beosin (Chengdu LianAn) // The 'increaseAllowance' function, 'msg.sender' increases the allowance which 'msg.sender' allowed to 'spender'.
    function increaseAllowance(address spender, uint addedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue)); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to increase the allowance between two addresses.
        return true;
    }
    // Beosin (Chengdu LianAn) // The 'decreaseAllowance' function, 'msg.sender' decreases the allowance which 'mag.sender' allowed to 'spender'.
    function decreaseAllowance(address spender, uint subtractedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC20: decreased allowance below zero")); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to decrease the allowance between two addresses.
        return true;
    }
    // Beosin (Chengdu LianAn) // The internal function '_transfer' is defined to do tranasfer operation.
    function _transfer(address sender, address recipient, uint amount) internal {
        require(sender != address(0), "ERC20: transfer from the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'sender'.
        require(recipient != address(0), "ERC20: transfer to the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'recipient'. Avoid losing transferred tokens.

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance"); // Beosin (Chengdu LianAn) // Alter the token balance of 'sender'.
        _balances[recipient] = _balances[recipient].add(amount); // Beosin (Chengdu LianAn) // Alter the token balance of 'recipient'.
        emit Transfer(sender, recipient, amount); // Beosin (Chengdu LianAn) // Trigger the event 'Transfer'.
    }
    // Beosin (Chengdu LianAn) // The internal function '_mint' is defined to do mint operation.
    function _mint(address account, uint amount) internal {
        require(account != address(0), "ERC20: mint to the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'account'.
        require(_totalSupply.add(amount) < maxSupply, "ERC20: cannot mint over max supply"); // Beosin (Chengdu LianAn) // Require that the total token supply after this mint should be less than the mint cap.

        _totalSupply = _totalSupply.add(amount); // Beosin (Chengdu LianAn) // Update the total token supply.
        _balances[account] = _balances[account].add(amount); // Beosin (Chengdu LianAn) // Alter the token balance of 'account'.
        emit Transfer(address(0), account, amount); // Beosin (Chengdu LianAn) // Trigger the event 'Transfer'.
    }
    // Beosin (Chengdu LianAn) // The internal function '_burn' is defined to do burn operation.
    function _burn(address account, uint amount) internal {
```

```
        require(account != address(0), "ERC20: burn from the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'account'.

        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance"); // Beosin (Chengdu LianAn) // Alter the token balance of 'account'.
        _totalSupply = _totalSupply.sub(amount); // Beosin (Chengdu LianAn) // Update the total token supply.
        emit Transfer(account, address(0), amount); // Beosin (Chengdu LianAn) // Trigger the event 'Transfer'.
    }
    // Beosin (Chengdu LianAn) // The internal function '_approve' is defined to do approve operation.
    function _approve(address owner, address spender, uint amount) internal {
        require(owner != address(0), "ERC20: approve from the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'owner'.
        require(spender != address(0), "ERC20: approve to the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'spender'.

        _allowances[owner][spender] = amount; // Beosin (Chengdu LianAn) // The allowance which 'owner' allowed to 'spender' is set to 'amount'.
        emit Approval(owner, spender, amount); // Beosin (Chengdu LianAn) // Trigger the event 'Approval'.
    }
}

contract ERC20Detailed is IERC20 {
    string private _name; // Beosin (Chengdu LianAn) // Declare the variable '_name' for storing the token name.
    string private _symbol; // Beosin (Chengdu LianAn) // Declare the variable '_symbol' for storing the token symbol.
    uint8 private _decimals; // Beosin (Chengdu LianAn) // Declare the variable '_decimals' for storing the token decimals.
     // Beosin (Chengdu LianAn) // Constructor, initialize the basic token information.
    constructor (string memory name, string memory symbol, uint8 decimals) public {
        _name = name;
        _symbol = symbol;
        _decimals = decimals;
    }
    // Beosin (Chengdu LianAn) // The function 'name' is defined to query the token name.
    function name() public view returns (string memory) {
        return _name;
    }
    // Beosin (Chengdu LianAn) // The function 'symbol' is defined to query the token symbol.
    function symbol() public view returns (string memory) {
        return _symbol;
    }
    // Beosin (Chengdu LianAn) // The function 'decimals' is defined to query the token decimals.
    function decimals() public view returns (uint8) {
        return _decimals;
    }
}
// Beosin (Chengdu LianAn) // The SafeMath library declares these functions for safe mathematical
```

```
operations.
library SafeMath {
    function add(uint a, uint b) internal pure returns (uint) {
        uint c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }
    function sub(uint a, uint b) internal pure returns (uint) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }
    function sub(uint a, uint b, string memory errorMessage) internal pure returns (uint) {
        require(b <= a, errorMessage);
        uint c = a - b;

        return c;
    }
    function mul(uint a, uint b) internal pure returns (uint) {
        if (a == 0) {
            return 0;
        }

        uint c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }
    function div(uint a, uint b) internal pure returns (uint) {
        return div(a, b, "SafeMath: division by zero");
    }
    function div(uint a, uint b, string memory errorMessage) internal pure returns (uint) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint c = a / b;

        return c;
    }
}
// Beosin (Chengdu LianAn) // The Address library declares the function 'isContract' to check whether the specified address 'account' is contract address.
library Address {
    function isContract(address account) internal view returns (bool) {
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != 0x0 && codehash != accountHash);
    }
```

```
}
// Beosin (Chengdu LianAn) // SafeERC20 library, its internal functions are used for safe external TRC20 contract transfer related call operations.
library SafeERC20 {
    using SafeMath for uint; // Beosin (Chengdu LianAn) // Use the SafeMath library for mathematical operation. Avoid integer overflow/underflow.
    using Address for address; // Beosin (Chengdu LianAn) // Use the function 'isContract' of Address library to check whether the specified address is contract address.

    function safeTransfer(IERC20 token, address to, uint value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    function safeApprove(IERC20 token, address spender, uint value) internal {
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }
    function callOptionalReturn(IERC20 token, bytes memory data) private {
        require(address(token).isContract(), "SafeERC20: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");

        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}

contract MFI is ERC20, ERC20Detailed {
  using SafeERC20 for IERC20; // Beosin (Chengdu LianAn) // Use the SafeERC20 library for safe external TRC20 contract calling.
  using Address for address; // Beosin (Chengdu LianAn) // Use the function 'isContract' of Address library to check whether the specified address is contract address.
  using SafeMath for uint; // Beosin (Chengdu LianAn) // Use the SafeMath library for mathematical operation. Avoid integer overflow/underflow.


  address public governance; // Beosin (Chengdu LianAn) // Declare the variable 'governance' for storing the governance management address.
```

```
  mapping (address => bool) public minters; // Beosin (Chengdu LianAn) // Declare the mapping variable 'minters' for storing the minter permission status of corresponding address.
  // Beosin (Chengdu LianAn) // Constructor, initialize the initial governance management address and execute the constructor of contract 'ERC20Detailed'.
  constructor () public ERC20Detailed("MoonPool.finance", "MFI", 18) {
      governance = msg.sender;
  }
  // Beosin (Chengdu LianAn) // The function 'mint' is defined to issue tokens to the specified 'account'.
  function mint(address account, uint amount) public {
      require(minters[msg.sender], "!minter"); // Beosin (Chengdu LianAn) // Permission check, require that the caller should have minter permission.
      _mint(account, amount); // Beosin (Chengdu LianAn) // Call the internal function '_mint' to issue tokens to specified address.
  }
  // Beosin (Chengdu LianAn) // The function 'setGovernance' is defined to update the governance management address.
  function setGovernance(address _governance) public {
      require(msg.sender == governance, "!governance"); // Beosin (Chengdu LianAn) // Permission check, require that the caller should be the current governance management address.
      governance = _governance; // Beosin (Chengdu LianAn) // Update the governance management address.
  }
  // Beosin (Chengdu LianAn) // The function 'addMinter' is defined to grant minter permission to '_minter'.
  function addMinter(address _minter) public {
      require(msg.sender == governance, "!governance"); // Beosin (Chengdu LianAn) // Permission check, require that the caller should be the current governance management address.
      minters[_minter] = true; // Beosin (Chengdu LianAn) // Change the minter permission status of '_minter' to true.
  }
  // Beosin (Chengdu LianAn) // The function 'removeMinter' is defined to remove the minter permission of '_minter'.
  function removeMinter(address _minter) public {
      require(msg.sender == governance, "!governance"); // Beosin (Chengdu LianAn) // Permission check, require that the caller should be the current governance management address.
      minters[_minter] = false; // Beosin (Chengdu LianAn) // Change the minter permission status of '_minter' to false.
  }
}
```

成都链安

BEOSIN

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com