BEOSIN
Blockchain Security

Smart contract security audit report

Audit Number：202101121116

Report Query Name: FC-LiquidityProvider

Smart Contract Info：

Smart Contract Name	Smart Contract Address	Smart Contract Address Link
LiquidityProvider	TSfVcQCYTV3WvpcFNGSSxmwU9ic98VQMCS	https://tronscan.org/#/contract/TSfVcQCYTV3WvpcFNGSSxmwU9ic98VQMCS/code

Start Date：2021.01.08

Completion Date：2021.01.12

Overall Result：Pass

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

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	Compiler Version Security	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		SafeMath Features	Pass
		require/assert Usage	Pass
		Gas Consumption	Pass
		Visibility Specifiers	Pass
		Fallback Usage	Pass
2	General Vulnerability	Integer Overflow/Underflow	Pass
		Reentrancy	Pass
		Pseudo-random Number Generator (PRNG)	Pass

		Transaction-Ordering Dependence	Pass
		DoS (Denial of Service)	Pass
		Access Control of Owner	Pass
		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass
		tx.origin Usage	Pass
		Replay Attack	Pass
		Overriding Variables	Pass
3	Business Security	Business Logics	Pass
		Business Implementations	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 project FC-LiquidityProvider, including Coding Standards, Security, and Business Logic. **The FC-LiquidityProvider project passed all audit items. The overall result is Pass (Distinction). The smart contract is able to function properly.** Immediate update is required to mitigate security risks. The smart contract is able to function properly only the possible operations mentioned above should be noticed.

Audit Contents:

1. Coding Conventions

Check the code style that does not conform to Solidity code style.

1.1 Compiler Version Security

- Description: Check whether the code implementation of current contract contains the exposed solidity compiler bug.
- Result: Pass

1.2 Deprecated Items

- Description: Check whether the current contract has the deprecated items.
- Result: Pass

1.3 Redundant Code

- Description: Check whether the contract code has redundant codes.
- Result: Pass

1.4 SafeMath Features

- Description: Check whether the SafeMath has been used. Or prevents the integer overflow/underflow in mathematical operation.
- Result: Pass

1.5 require/assert Usage

- Description: Check the use reasonability of 'require' and 'assert' in the contract.
- Result: Pass

1.6 Gas Consumption

- Description: Check whether the gas consumption exceeds the block gas limitation.
- Result: Pass

1.7 Visibility Specifiers

- Description: Check whether the visibility conforms to design requirement.
- Result: Pass

1.8 Fallback Usage

- Description: Check whether the Fallback function has been used correctly in the current contract.
- Result: Pass

2. General Vulnerability

Check whether the general vulnerabilities exist in the contract.

2.1 Integer Overflow/Underflow

- Description: Check whether there is an integer overflow/underflow in the contract and the calculation result is abnormal.
- Result: Pass

2.2 Reentrancy

- Description: An issue when code can call back into your contract and change state, such as withdrawing ETH.
- Result: Pass

2.3 Pseudo-random Number Generator (PRNG)

- Description: Whether the results of random numbers can be predicted.
- Result: Pass

2.4 Transaction-Ordering Dependence

- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: Pass

2.5 DoS (Denial of Service)

- Description: Whether exist DoS attack in the contract which is vulnerable because of unexpected reason.
- Result: Pass

2.6 Access Control of Owner

- Description: Whether the owner has excessive permissions, such as malicious issue, modifying the balance of others.
- Result: Pass

2.7 Low-level Function (call/delegatecall) Security

- Description: Check whether the usage of low-level functions like call/delegatecall have vulnerabilities.
- Result: Pass

2.8 Returned Value Security

- Description: Check whether the function checks the return value and responds to it accordingly.
- Result: Pass

2.9 tx.origin Usage

- Description: Check the use secure risk of 'tx.origin' in the contract. In this project, the contract
- Result: Pass

2.10 Replay Attack

- Description: Check the weather the implement possibility of Replay Attack exists in the contract.
- Result: Pass

2.11 Overriding Variables

- Description: Check whether the variables have been overridden and lead to wrong code execution.
- Result: Pass

3. Business Security

3.1 Business analysis of Contract LiquidityProvider

(1) Add liquidity function

- Description: As shown in the figure below, the contract implements the *addLiquidity* function to add liquidity, requiring the caller to be the owner of the contract. The *tokenAmount* entered must not exceed the balance of the token in the contract. Then, liquidity is injected by calling the *addLiquidity* function of the input LP token address.

```
20    function addLiquidity(address lpAddr, address tokenAddr, uint256 tokenAmount) public payable returns (bool) {
21        require(msg.sender == _owner, "sender is not owner");
22
23        IERC20 token = IERC20(tokenAddr);
24        uint256 tokenBalance = token.balanceOf(address(this));
25        require(tokenBalance >= tokenAmount, "token is not sufficient");
26        token.approve(lpAddr, tokenAmount);
27
28        IJustswapExchange exchange = IJustswapExchange(lpAddr);
29        exchange.addLiquidity.value(msg.value)(1, tokenAmount, now + 60);
30
31        return true;
32    }
```

Figure 1 source code of *addLiquidity*

- Related functions: *addLiquidity*

- Result: Pass

(2) Withdraw function

- Description: As shown in the figure below, the contract implements the withdraw function to extract all the specified tokens in this contract to the specified address, requiring the caller to be the owner of the contract. If the input *cntr* address is 0 address, all TRX in this contract will be withdrawn; if it is not 0 address, the specified TRC20 tokens will be withdrawn.

```
34    function withdraw(address cntr, address payable recipient) public returns (bool) {
35        require(msg.sender == _owner, "sender is not owner");
36
37        if (cntr == address(0)) {
38            uint256 balance = address(this).balance;
39            if (balance > 0) {
40                recipient.transfer(balance);
41                return true;
42            } else {
43                return false;
44            }
45        } else {
46            IERC20 token = IERC20(cntr);
47            uint256 balance = token.balanceOf(address(this));
48            if (balance > 0) {
49                token.transfer(recipient, balance);
50                return true;
51            } else {
52                return false;
53            }
54        }
55    }
56 }
```

Figure 2 source code of *withdraw*

- Related functions: *withdraw*

- Result: Pass

4. Conclusion

Beosin(ChengduLianAn) conducted a detailed audit on the design and code implementation of the smart contracts project FC-LiquidityProvider. The problems found by the audit team during the audit process have been notified to the project party and fixed, the overall audit result of the FC-LiquidityProvider project's smart contract is **Pass**.

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com