SMART CONTRACT AUDIT REPORT

For

TronFactoryVIP

Contract Address: TXDwN9YMqpfrvgycVWGzCZCf5U8ayKZY66

Prepared By: Kishan Patel

Prepared For: TronFactory.VIP

Prepared on: 30/01/2021

Table of Content

• Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

• Overview of the audit

The project has 1 file. It contains approx 334 lines of Solidity code. All the functions and state variables are well commented using the natspec documentation, but that does not create any vulnerability.

• Attacks made to the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices.

• Over and under flows

An overflow happens when the limit of the type variable uint256, 2 ** 256, is exceeded. What happens is that the value resets to zero instead of incrementing more. On the other hand, an underflow happens when you try to subtract 0 minus a number bigger than 0. For example, if you subtract 0 - 1 the result will be = 2 ** 256 instead of -1. This is quite dangerous.

This contract does check for overflows and underflows by using OpenZeppelin's SafeMath to mitigate this attack, but all the functions have strong validations, which prevented this attack.

• Short address attack

If the token contract has enough amount of tokens and the buy function doesn't check the length of the address of the sender, the Tron's virtual machine will just add zeros to the transaction until the address is complete.

Although this contract **is not vulnerable** to this attack, but there are some point where users can mess themselves due to this (Please see below). It is highly recommended to call functions after checking validity of the address.

• Visibility & Delegate call

It is also known as, The Parity Hack, which occurs while misuse of Delegate call.

No such issues found in this smart contract and visibility also properly addressed. There are some places where there is no visibility defined. Smart Contract will assume "Public" visibility if there is no visibility defined. It is good practice to explicitly define the visibility, but again, the contract is not prone to any vulnerability due to this in this case.

• Reentrancy / TheDAO hack

Reentrancy occurs in this case: any interaction from a contract (A) with another contract (B) and any transfer of Tron hands over control to that contract (B).

This makes it possible for B to call back into A before this interaction is completed.

Use of "require" function in this smart contract mitigated this vulnerability.

• Forcing Tron to a contract

While implementing "selfdestruct" in smart contract, it sends all the tron to the target address. Now, if the target address is a contract address, then the fallback function of target contract does not get called. And thus Hacker can bypass the

"Required" conditions. Here, the Smart Contract's balance has never been used as guard, which mitigated this vulnerability.

• Good things in smart contract

• SafeMath library:-

o You are using SafeMath library it is a good thing. This protects you from underflow and overflow attacks.

```
301
302 library SafeMath {
303     function mul(uint256 a, uint256 b) internal pure returns (uint256) {
304         if (a == 0) {
305             return 0;
306         }
```

• Good required condition in functions:-

o Here you are checking that contract is released and msg.value is bigger or equal to minDepositSize(100000000).

```
94
95     function deposit(address _affAddr) public payable {
96         require(now >= releaseTime, "not launched yet!");
97         collect(msg.sender);
98         require(msg.value >= minDepositSize);
```

o Here you are checking that msg.sender has more than 0 arrReward to call this function.

```
134
135     function withdraw_referral() public {
136         require(players[msg.sender].affRewards > 0);
137
138         transferReferral(msg.sender, players[msg.sender].affRewards);
139     }
```

o Here you are checking that msg.sender has more than 0 interestProfit to call this function.

```
141     function withdraw() public {
142         collect(msg.sender);
143         require(players[msg.sender].interestProfit > 0);
144
145         transferPayout(msg.sender, players[msg.sender].interestProfit);
```

o Here you are checking that contract balance is bigger or equal to depositeAmount.

```
148     function reinvest() public {
149         collect(msg.sender);
150         Player storage player = players[msg.sender];
151         uint256 depositAmount = player.interestProfit;
152         require(address(this).balance >= depositAmount);
```

o Here you are checking that user is invested or interacted with your contract then and then they can call this function.

```
263
264     function getProfit(address _addr) public view returns (uint256) {
265         address playerAddress = _addr;
266         Player storage player = players[playerAddress];
267         require(player.time > 0, "player time is 0");
268
```

- Critical vulnerabilities found in the contract

=> No Critial vulnerabilities found

- Medium vulnerabilities found in the contract

=> No Medium vulnerabilities found

- Low severity vulnerabilities found

  - 7.2: Short address attack:-

=> This is not big issue in solidity, because now a days is increased In the new solidity version. But it is good practice to Check for the short address.

=> After updating the version of solidity it's not mandatory.

=> In some functions you are not checking the value of address parameter.

=> Your some logic is depend on not a address so please check and update your code as per your requirements.

- Function:- register, setRefCount (‘_addr’, ‘_addAddr’)

```
73
74      function register(address _addr, address _affAddr) private {
75          Player storage player = players[_addr];
76
77          player.affFrom = _affAddr;
78          players[_affAddr].td_team = players[_affAddr].td_team.add(1);
79      }
82
83      function setRefCount(address _addr, address _affAddr) private {
84          Preferral storage preferral = preferals[_addr];
85          preferral.player_addr = _addr;
86          address _affAddr1 = _affAddr;
```

  - It's necessary to check the addresses value of "_addr","_addAddr". Because here you are passing whatever variable comes in "_addr","_addAddr" addresses from outside.

- **Function: - collect (‘ addr’)**

```
166
167     function collect(address _addr) internal {
168         Player storage player = players[_addr];
169
170         uint256 secPassed = now.sub(player.time);
171         if (secPassed > 864000 && player.time > 0) {
```

  - It's necessary to check the address value of "_addr". Because here you are passing whatever variable come in "_addr" address from outside.

- **Function: - getProfit (‘ addr’)**

```
263
264     function getProfit(address _addr) public view returns (uint256) {
265         address playerAddress = _addr;
266         Player storage player = players[playerAddress];
267         require(player.time > 0, "player time is 0");
268
```

  - It's necessary to check the address value of "_addr". Because here you are passing whatever variable comes in "_addr" address from outside.

7.3: Unchecked return value or response:-

=> I have found that you are transferring fund to address using a transfer method.

=> It is always good to check the return value or response from a function call. Because some time this transfer failed and your code run successfully.

=> Here are some functions where you forgot to check a response.

=> I suggest, if there is a possibility then please check the response.

- Function: - deposit

```
128         uint256 feed2earn =
129             depositAmount.mul(devCommission).mul(4).div(commissionDiv
130
131         feed1.transfer(feed1earn);
132         feed2.transfer(feed2earn);
133     }
```

  - Here you are calling transfer method 2 times. It is good to check that the transfer is successfully done or not.

- **Function: - reinvest**

```
646
647     function safedefitokentestTransferUpgrade(address to, uint8 level) internal {
648         uint256 defitokentestBal = BalanceOfTokenInContract();
649         if(defitokentestBal >= convertToToken(RewardToken[level]) && level>1){
650         if(convertToToken(101000000) > UpgradeTokenDistributed){
651             TRC20Interface(Tokenaddress).transfer(to, convertToToken(RewardToken[level]
652         UpgradeTokenDistributed += convertToToken(RewardToken[level]);
653         }
```

  - Here you are calling transfer method a time. It is good to check that the transfer is successfully done or not.

- **Function: - withdraw**

```
160         uint256 feed2earn =
161             depositAmount.mul(devCommission).mul(4).div(commissionDivisor);
162
163         feed1.transfer(feed1earn);
164         feed2.transfer(feed2earn);
165     }
```

  - Here you are calling transfer method 2 times. It is good to check that the transfer is successfully done or not.

- **Function: - transferReferral**

```
198                 Player storage player = players[_receiver];
199
200                 player.payoutSum = player.payoutSum.add(payout);
201                 msg.sender.transfer(payout);
202             }
203         }
```

- Here you are calling transfer method 1 times. It is good to check that the transfer is successfully done or not.

Function: - transferPayout

```
223
224                 uint256 withdraw_fee = (payout * 8) / 100;
225                 payout = payout - withdraw_fee;
226                 owner.transfer(withdraw_fee);
227                 msg.sender.transfer(payout);
228             }
```

- Here you are calling transfer method 2 times. It is good to check that the transfer is successfully done or not.

• Summary of the Audit

Overall the code is well and performs well.

Please try to check the address and value of token externally before sending to the solidity code.

Our final recommendation would be to pay more attention to the visibility of the functions , hardcoded address and mapping since it's quite important to define who's supposed to executed the functions and to follow best practices regarding the use of assert, require etc. (which you are doing ;)).

- Note: Please focus on a version use latest, check the response of transfer method, and check addresses.
- I have seen that a developer is using block's timestamp and now method so, I like to tell you that write smart contracts with the notion that block values are not precise, and the use of them can lead to unexpected effects.