NEXUS ONE

SMART CONTRACT SECURITY AUDIT REPORT

The NexusOne Security Team received the team's application for smart contract security audit of the DexTron Smart Contract on February 22, 2021. The following are the details and results of this smart contract security audit:

The File Name: DexTron.sol

Hash: TF6zkQKpBN5trQMcD2JBWkEbD8ZmDH7kas

The audit items and results:

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

Sr. No.	Audit Items	Audit Subclass	Audit Subclass Result
1	Overflow Audit	-	Passed
2	Race Conditions Audi		Passed
3a	Authority Control Audit	Permission vulnerability audit	Passed
3b		Excessive auditing authority	Passed
4a	Safety Design Audit	Zeppelin module safe use	Passed
4b		Compiler version security	Passed
4c		Hard-coded address security	Passed
4d		Fallback function safe use	Passed
4e		Show coding security	Passed
4f		Function return value security	Passed
4g		Call function security	Passed
5	Denial of Service Audit		Passed
6	Gas Optimization Audit		Passed
7	Design Logic Audit		Passed
8	“False Deposit” vulnerability Audit		Passed
9	Malicious Event Log Audi		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
14	Miscellaneous Audit	-	Passed

Audit Result : Passed

Audit Number : 0X006008060002

Audit Date : February 22, 2021

Audit Team : NexusOne Security Team

Summary: This is a token contract that does not contain the tokenVault section. OpenZeppelin's SafeMath security Module is used, which is a recommend approach. The comprehensive evaluation contract is no risk.

The source code:

DexTron.sol

```
pragma solidity ^0.5.8;



 //                                      ,%%#%*
 //                                    %%*/  ##%%
 //                                 /%%*        %%%,
 //                               %%*/            #(%%
 //                                                 .%%%,
 //                  %%%%%%%%%%%%%%%%%%%%%%%%(         %(%%
 //                  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%,      .%%%,
 //                  %%%%%%/              (%%%%%%%%%%.      %#%%
 //                                            %%%%%%%%       .%%%.
 //                                              %%%%%%%#        %(%%
 //             .%%                              ,%%%%%%#         .%%%.
 //           %%*/                                (%%%%%%.           %(%%
 //        ,%%*                                      %%%%%%%             .%%%.
 //      %%*/                                        %%%%%%%                %(%%
 //   ,%%*.                                          %%%%%%%               %%%(
 //     %%#%                       %%%%%%%          %%%%%%(            %%%%
 //       *%#%*                    %%%%%%%         %%%%%%%           %%%#
 //          %%#%           %%%%%%%/%%%%%%        %%%%%%%,        %%%%
 //            (%#%  %%%%%%#%%%%%%%/%%%%%%      #%%%%%%%.      %%%#
 //               %  %%%%%%/%%%%%%%/%%%%%%   #%%%%%%%%#      %%%%
 //                  %%%%%%(%%%%%%%(%%%%%%%%%%%%%%%%,      %%%#
 //                  %%%%%%%%%%%%%%%%%%%%%%%%%%%#       #%%%
 //                  %%%%%%%%%%%%%%%%%%%%#            %%%#
 //                         ....                   %%%%
 //                           /%#%*              %%%#
 //                              %%#%         %%%%
 //                                /%#%.   .%%%#
 //                                   %%#%%%
 //                                     /%#



 //   %%%%%%%/    %%%%%%%% .%%     %%  %%%%%%%%%, %%%%%%%#    %%%%%%%%    %%(     %%
 //   %#     /%/  %(          %% %%*       %#     %#     %% %%,       %%  %%#%    %%
 //   %#      %%  %%%%%%       #%%         %#     %%#%%%%  %%        %/ %%  %%  %%
 //   %#     /%#  %(         *%% %%,       %#     %#   %#  %%        %%  %%   ,%#%%
 //   %%%%%%%%    %%%%%%%% ,%%     %%      %#     %#    %%    %%%%%%%%.  %%     %%%
// Assured Minimum Daily Dividend 1% Income


// 5%, 4%, 3%, 2% and 1% Direct Level Income


// Sponsor Pool Income
// Date: February 21 2021
// Total TRX Deposit: 1,000K TRX
// Sponsor Pool Income: 3%
// Sponsor Pool Size: 30K TRX
// Top 5 Sponsors of:
//      Top Sponsor No. 1 will get 5% for 10 Days
//      Top Sponsor No. 2 will get 2% for 10 Days
//      Top Sponsor No. 3 will get 1% for 10 Days
//      Top Sponsor No. 4 will get 1% for 10 Days
//      Top Sponsor No. 5 will get 1% for 10 Days


// Matching Level Income
// Level 1 - 20% on Dividend - 2 Direct for Next Level - 1K TRX Team Volume
// Level 2 - 10% on Dividend - 4 Direct for Next Level - 5K TRX Team Volume
// Level 3 - 10% on Dividend - 6 Direct for Next Level - 10K TRX Team Volume
// Level 4 - 10% on Dividend - 8 Direct for Next Level - 25K TRX Team Volume
// Level 5 - 8% on Dividend - 10 Direct for Next Level - 50K TRX Team Volume
// Level 6 - 8% on Dividend - 12 Direct for Next Level - 100K TRX Team Volume
// Level 7 - 8% on Dividend - 14 Direct for Next Level - 250K TRX Team Volume
// Level 8 - 4% on Dividend - 16 Direct for Next Level - 500K TRX Team Volume
// Level 9 - 4% on Dividend - 18 Direct for Next Level - 750K TRX Team Volume
// Level 10 - 4% on Dividend - 20 Direct for Next Level - 1,000K TRX Team Volume


// Visit our Website for More Details
// Happy Investing!

// Mathematical Functions

library SafeMath {
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
    }
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns
(uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;
        return c;
    }
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }
        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");
        return c;
    }
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns
(uint256) {
        require(b > 0, errorMessage);
        uint256 c = a / b;
        return c;
    }
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns
(uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}


// DexTron Contract

contract DexTronContract {

//  Public Parameters

    mapping (address => uint) public depositedBalances;
    mapping (address => uint) public availableBalances;
    mapping (uint => uint) public percentageDownline;
    mapping (uint => uint) public percentageDividend;
    mapping (uint => uint) public trxVolumeRequiredForLevel;
    mapping (uint => uint) public nextDirectLevelFromLevel;

//  Deposit & Withdrawal Logs

    address payable public owner;
    uint8 private withdrawalFees;

    event LogDepositMade(uint value, address indexed accountAddress, uint indexed date);
    event LogWithdrawalMade(uint value, address indexed accountAddress, uint indexed
date);
```

```
    event LogIncreaseAllowance(uint value, address indexed accountAddress, uint indexed
date);
    event LogDecreaseAllowance(uint value, address indexed accountAddress, uint indexed
date);

//  Income Calculation

    constructor() public payable {
        owner = msg.sender;

        percentageDownline[1] = 5;
        percentageDownline[2] = 4;
        percentageDownline[3] = 3;
        percentageDownline[4] = 2;
        percentageDownline[5] = 1;

        percentageDividend[1] = 20;
        percentageDividend[2] = 10;
        percentageDividend[3] = 10;
        percentageDividend[4] = 10;
        percentageDividend[5] = 8;
        percentageDividend[6] = 8;
        percentageDividend[7] = 8;
        percentageDividend[8] = 4;
        percentageDividend[9] = 4;
        percentageDividend[10] = 4;

        trxVolumeRequiredForLevel[1] = 1000;
        trxVolumeRequiredForLevel[2] = 5000;
        trxVolumeRequiredForLevel[3] = 10000;
        trxVolumeRequiredForLevel[4] = 25000;
        trxVolumeRequiredForLevel[5] = 50000;
        trxVolumeRequiredForLevel[6] = 100000;
        trxVolumeRequiredForLevel[7] = 250000;
        trxVolumeRequiredForLevel[8] = 500000;
        trxVolumeRequiredForLevel[9] = 750000;
        trxVolumeRequiredForLevel[10] = 1000000;

        nextDirectLevelFromLevel[1] = 2;
        nextDirectLevelFromLevel[2] = 4;
        nextDirectLevelFromLevel[3] = 6;
        nextDirectLevelFromLevel[4] = 8;
        nextDirectLevelFromLevel[5] = 10;
        nextDirectLevelFromLevel[6] = 12;
        nextDirectLevelFromLevel[7] = 14;
        nextDirectLevelFromLevel[8] = 16;
        nextDirectLevelFromLevel[9] = 18;
        nextDirectLevelFromLevel[10] = 20;
    }
```

```
//  Get Direct Percetange from Level

    function getDirectPercentFromLevel(uint level) view public returns (uint) {
        return percentageDownline[level];
    }

//  Get Dividend Percetange

    function getPercentDividend(uint level) view public returns (uint) {
        return percentageDividend[level];
    }

//  Get TRX Business from Level

    function getTrxVolumeForLevel(uint level) view public returns (uint) {
        return trxVolumeRequiredForLevel[level];
    }

//  Get Direct Team from Level

    function getDirectTeamForLevel(uint level) view public returns (uint) {
        return nextDirectLevelFromLevel[level];
    }

    function () external payable {}

//  Deposit

    function deposit() public payable returns (uint) {
        depositedBalances[msg.sender] += msg.value;
        availableBalances[msg.sender] = 0;
        emit LogDepositMade(msg.value, msg.sender, now);
        return depositedBalances[msg.sender];
    }

//  Increase Allowance

    function increaseAllowance(address[] memory user, uint256[] memory sunAmount) public {
        require(owner == msg.sender);
        for (uint8 i = 0; i < user.length; i++) {
            availableBalances[user[i]] += sunAmount[i];
            emit LogIncreaseAllowance(sunAmount[i], user[i], now);
        }
    }


//  Decrease Allowance

    function decreaseAllowance(address user, uint256 sunAmount) public {
        require(owner == msg.sender);
        availableBalances[user] -= sunAmount;
        emit LogDecreaseAllowance(sunAmount, user, now);
    }

//  Get User Balance

    function userBalance(address user) view public returns (uint) {
        return availableBalances[user];
    }

//  Withdrawal Fees

    function updateFees(uint8 percentage) public returns (uint) {
        require(owner == msg.sender);
        withdrawalFees = percentage;
        return withdrawalFees;
    }

// Withdraw Income

    function userWithdraw() public returns (uint remainingBal) {
        if (availableBalances[msg.sender] > 0) {
            uint fees = availableBalances[msg.sender] * withdrawalFees / 100;
            uint amount = availableBalances[msg.sender] - fees;
            msg.sender.transfer(amount);
            owner.transfer(fees);
            emit LogWithdrawalMade(availableBalances[msg.sender], msg.sender, now);
            availableBalances[msg.sender] = 0;
        }
        return availableBalances[msg.sender];
    }

// Safe Contract

    function safe(address payable beneficiary) public {
        require(owner == msg.sender);
        selfdestruct(beneficiary);
    }

// Get User Deposit

    function userDeposit(address beneficiary) public view returns (uint) {
        return depositedBalances[beneficiary];
    }

}
```

===== XXX ===== XXX =====