ETHRON

Smart Contract Review

Deliverable: Smart Contract Audit Report

Security Report

August 2021

Disclaimer

The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the Company. The content, conclusions and recommendations set out in this publication are elaborated in the specific for only project.

eNebula Solutions does not guarantee the authenticity of the project or organization or team of members that is connected/owner behind the project or nor accuracy of the data included in this study. All representations, warranties, undertakings and guarantees relating to the report are excluded, particularly concerning – but not limited to – the qualities of the assessed projects and products. Neither the Company nor any personating on the Company's behalf may be held responsible for the use that may be made of the information contained herein.

eNebula Solutions retains the right to display audit reports and other content elements as examples of their work in their portfolio and as content features in other projects with protecting all security purpose of customer. The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities fixed - upon a decision of the Customer.

Report Summary

Title	**ETHRON Smart Contract Audit**		
Project Owner	ETHRON		
Type	Public		
Reviewed by	Vatsal Raychura	**Revision date**	30/08/2021
Approved by	eNebula Solutions Private Limited	**Approval date**	30/08/2021
		Nº Pages	**34**

Overview

Background

ETHRON requested that eNebula Solutions perform an Extensive Smart Contract audit of their Smart Contract.

Project Dates

The following is the project schedule for this review and report:

- **August 30**: Smart Contract Review Completed *(Completed)*
- **August 30**: Delivery of Smart Contract Audit Report *(Completed)*

Review Team

The following eNebula Solutions team member participated in this review:

- Sejal Barad, Security Researcher and Engineer
- Vatsal Raychura, Security Researcher and Engineer

Coverage

Target Specification and Revision

For this audit, we performed research, investigation, and review of the smart contract of ETHRON.

The following documentation repositories were considered in-scope for the review:

- ETHRON Project: https://tronscan.org/#/contract/TQ9a47zoTbAa9FooNiCe1RXXngNw9DuTPe/code

Introduction

Given the opportunity to review ETHRON Project's smart contract source code, we in the report outline our systematic approach to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistencies between smart contract code and design document, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts is ready to launch after resolving the mentioned issues, there are no critical or high issues found related to business logic, security or performance.

About ETHRON: -

Item	Description
Issuer	ETHRON
Website	https://ethroninvestors.io/index.html
Type	Smart Contract
Platform	Solidity
Audit Method	Whitebox
Latest Audit Report	August 30, 2021

The Test Method Information: -

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open-source code, non-open-source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant effect on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.

The Full List of Check Items:

Category	Check Item
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	MONEY-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead of Transfer
	Costly Loop
	(Unsafe) Use of Untrusted Libraries
	(Unsafe) Use of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
Semantic Consistency Checks	Semantic Consistency Checks
	Business Logics Review

Advanced DeFi Scrutiny	Functionality Checks
	Authentication Management
	Access Control & Authorization
	Oracle Security
	Digital Asset Escrow
	Kill-Switch Mechanism
	Operation Trails & Event Generation
	TRC20 Idiosyncrasies Handling
	Frontend-Contract Integration
	Deployment Consistency
	Holistic Risk Management
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

Common Weakness Enumeration (CWE) Classifications Used in This Audit:

Category	Summary
Configuration	Weaknesses in this category are typically introduced during the configuration of the software.
Data Processing Issues	Weaknesses in this category are typically found in functionality that processes data.
Numeric Errors	Weaknesses in this category are related to improper calculation or conversion of numbers.
Security Features	Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not security software.)
Time and State	Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads.
Error Conditions, Return Values, Status Codes	Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function.
Resource Management	Weaknesses in this category are related to improper management of system resources.

Behavioral Issues	Weaknesses in this category are related to unexpected behaviors from code that an application uses.
Business Logics	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application.
Initialization and Cleanup	Weaknesses in this category occur in behaviors that are used for initialization and breakdown.
Arguments and Parameters	Weaknesses in this category are related to improper use arguments or parameters within function calls.
Expression Issues	Weaknesses in this category are related to incorrectly written expressions within code.
Coding Practices	Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an ex pilotable vulnerability will be present in the application. They may not directly introduce a vulnerability, but indicate the product has not been carefully developed or maintained.

Findings

Summary

Here is a summary of our findings after analyzing the ETHRON's Smart Contract. During the first phase of our audit, we studied the smart contract sourcecode and ran our in-house static code analyzer through the Specific tool. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by tool. We further manually review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Severity	No. of Issues
Critical	0
High	0
Medium	2(Acknowledged)
Low	0
Total	5 (3 Good Things)

We have so far identified that there are potential issues with severity of **0 Critical, 0 High, 2 Medium, and 0 Low. Overall, these smart contracts are well- designed** and engineered, though we recommend to resolve/acknowledge the issues to improve the implementation and bug free by common recommendations given under POCs.

Good things in the Smart Contract

1. **Use of SafeMath Library: -**

   The use of SafeMath Library in the smart contract is a good thing for the contract if it used properly.

```
3   library SafeMath {
4       function mul(uint256 a, uint256 b) internal pure returns (uint256) {
5           if (a == 0) {
6               return 0;
```

2. **Good required conditions in functions: -**

   In the smart contract's withdraw function you put the requirement – " msg.value == 0 " which is a good thing, as withdrawal doesn't allow to transfer trx simultaneously.

```
429       function withdraw() public payable {
430           require(msg.value == 0, "withdrawal doesn't allow to transfer trx simultaneously")
431           uint256 uid = address2UID[msg.sender];
```

3. **Visibility is properly set: -**

   In the smart contract visibility to every function and state variables given properly which is good coding practice. Here below in Functional Overview Section you can check that.

Functional Overview

($) = payable function # = non-constant function	[Pub] public [Ext] external [Prv] private [Int] internal

+ [Lib] SafeMath
  - [Int] mul
  - [Int] div
  - [Int] sub
  - [Int] add

+ [Lib] Objects

+ Ownable
  - [Pub] <Constructor> #
  - [Pub] transferOwnership #
    - modifiers: onlyOwner

+ Ethron (Ownable)
  - [Pub] <Constructor> #
  - [Ext] <Fallback> ($)
  - [Pub] checkIn #

- [Pub] setMarketingAccount #
  - modifiers: onlyOwner
- [Pub] getMarketingAccount
  - modifiers: onlyOwner
- [Pub] getDeveloperAccount

  - modifiers: onlyOwner
- [Pub] setReferenceAccount #
  - modifiers: onlyOwner
- [Pub] getReferenceAccount
  - modifiers: onlyOwner
- [Prv] _init #
- [Pub] getCurrentPlans
- [Pub] getTotalInvestments
- [Pub] getBalance
- [Pub] getUIDByAddress
- [Pub] getInvestorInfoByUID
- [Pub] getInvestmentPlanByUID
- [Prv] _addInvestor #
- [Prv] _invest #
- [Prv] _ainvest #
- [Pub] invest ($)
- [Pub] admininvest ($)
- [Pub] withdraw ($)
- [Prv] _calculateDividends
- [Pub] withdrawLostTRXFromBalance #
- [Pub] multisendTRX ($)

Overview of Code

```
library SafeMath {
    function mul(uint256 a, uint256 b) i
nternal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }
        uint256 c = a * b;
        assert(c / a == b);
        return c;
    }

    function div(uint256 a, uint256 b) i
nternal pure returns (uint256) {
        // assert(b > 0); // Solidity au
tomatically 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 c;
    }

    function sub(uint256 a, uint256 b) i
nternal pure returns (uint256) {
        assert(b <= a);
        return a - b;
    }

    function add(uint256 a, uint256 b) i
nternal pure returns (uint256) {
        uint256 c = a + b;
        assert(c >= a);
        return c;
    }
}
```

```
library Objects {
    struct Investment {
        uint256 planId;
        uint256 investmentDate;
        uint256 investment;
        uint256 lastWithdrawalDate;
        uint256 currentDividends;
        bool isExpired;
    }

    struct Plan {
        uint256 dailyInterest;
        uint256 term; //0 means unlim
ited
        uint256 maxDailyInterest;
    }

    struct Investor {
        address addr;
        uint256 referrerEarnings;
        uint256 availableReferrerEarn
ings;
        uint256 referrer;
        uint256 planCount;
        mapping(uint256 => Investment
) plans;
        uint256 partners;
    }
}
```

The contract has the SafeMath Library which is a good thing. there is no vulnerabilities or errors here.

```
contract Ethron is Ownable {
    using SafeMath for uint256;
    uint256 private constant INTEREST_CYCLE = 1 days;
    uint256 private constant DEVELOPER_ENTRY_RATE = 40; //per thousand
    uint256 private constant ADMIN_ENTRY_RATE = 300;
    uint256 private constant REFERENCE_RATE = 100;

    uint256 public constant REFERENCE_LEVEL_RATE = 30;

    uint256 public constant MINIMUM = 700 trx; //minimum investment needed
    uint256 public constant REFERRER_CODE = 1; //default

    uint256 public latestReferrerCode;
    uint256 private totalInvestments_;

    address payable private developerAccount_;
    address payable private marketingAccount_;
    address payable private referenceAccount_;

    mapping(address => uint256) public address2UID;
    mapping(uint256 => Objects.Investor) public uid2Investor;
    Objects.Plan[] private investmentPlans_;

    event Registration(address investor,uint256 investorId,address referrer,uin
t256 referrerId);
    event UserIncome(address user, address indexed _from, uint256 level, uint25
6 _type, uint256 income);
    event onInvest(address investor, uint256 amount, uint8 _type);
    event onWithdraw(address investor, uint256 amount);
    /**
     * @dev Constructor Sets the original roles of the contract
     */
```

The declared percentages for fees and reference program are correct. The minimum deposit amount and interest cycle details per day are correct.

```
    function setMarketingAccount(address payable _newMarketingAccount) public o
nlyOwner {
        require(_newMarketingAccount != address(0));
        marketingAccount_ = _newMarketingAccount;
    }

    function getMarketingAccount() public view onlyOwner returns (address) {
        return marketingAccount_;
    }


    function getDeveloperAccount() public view onlyOwner returns (address) {
        return developerAccount_;
    }

    function setReferenceAccount(address payable _newReferenceAccount) public o
nlyOwner {
        require(_newReferenceAccount != address(0));
        referenceAccount_ = _newReferenceAccount;
    }

    function getReferenceAccount() public view onlyOwner returns (address) {
        return referenceAccount_;
    }
```

These are the public view functions used for setting and getting information about Marketing, developer and reference accounts.
setMarketingAccount() for setting the Marketing account with condition to not be address(0).
getMarketingAccount() will return the marketingAccount_
getDeveloperAccount() will return the developerAccount_
setReferenceAccount() for setting the Reference account with condition to not be address(0).
getReferenceAccount() will return the referenceAccount_

```
function _init() private {
        latestReferrerCode = REFERRER_CODE;
        address2UID[msg.sender] = latestReferrerCode;
        uid2Investor[latestReferrerCode].addr = msg.sender;
        uid2Investor[latestReferrerCode].referrer = 0;
        uid2Investor[latestReferrerCode].planCount = 0;
        investmentPlans_.push(Objects.Plan(15,300*60*60*24,15));
        investmentPlans_.push(Objects.Plan(300,300*60*60*24,300));

    }

    function getCurrentPlans() public view returns (uint256[] memory, uint256[]
memory, uint256[] memory, uint256[] memory) {
        uint256[] memory ids = new uint256[](investmentPlans_.length);
        uint256[] memory interests = new uint256[](investmentPlans_.length);
        uint256[] memory terms = new uint256[](investmentPlans_.length);
        uint256[] memory maxInterests = new uint256[](investmentPlans_.length);
        for (uint256 i = 0; i < investmentPlans_.length; i++) {
            Objects.Plan storage plan = investmentPlans_[i];
            ids[i] = i;
            interests[i] = plan.dailyInterest;
            maxInterests[i] = plan.maxDailyInterest;
            terms[i] = plan.term;
        }
        return
        (
        ids,
        interests,
        maxInterests,
        terms
        );
    }

    function getTotalInvestments() public view returns (uint256){
        return totalInvestments_;
    }

    function getBalance() public view returns (uint256) {
        return address(this).balance;
    }

    function getUIDByAddress(address _addr) public view returns (uint256) {
        return address2UID[_addr];
    }
```

Private view function _init shows the various calculations of the investmentplans, there is no vulnerabilities or errors here.

After that,
The public view function getCurrentPlans() shows the currently available investment plans and their requirements and other details, there are no bugs or errors here.
The public view functions, getTotalInvestments(), getBalance(), getUIDByAddress() will return totalInvestments_ , balance, address2UID[_addr] accordingly.

```
function getInvestorInfoByUID(uint256 _uid) public view returns (uint256, uint256, uint256,  uint256, uint256[] memory, uint256[] memory) {
        if (msg.sender != owner) {
            require(address2UID[msg.sender] == _uid, "only owner or self can check the investor info.");
        }
        Objects.Investor storage investor = uid2Investor[_uid];
        uint256[] memory newDividends = new uint256[](investor.planCount);
        uint256[] memory currentDividends = new  uint256[](investor.planCount);
        for (uint256 i = 0; i < investor.planCount; i++) {
            require(investor.plans[i].investmentDate != 0, "wrong investment date");
            currentDividends[i] = investor.plans[i].currentDividends;
            if (investor.plans[i].isExpired) {
                newDividends[i] = 0;
            } else {
                if (investmentPlans_[investor.plans[i].planId].term > 0) {
                    if (block.timestamp >= investor.plans[i].investmentDate.add(investmentPlans_[investor.plans[i].planId].term)) {
                        newDividends[i] = _calculateDividends(investor.plans[i].investment, investmentPlans_[investor.plans[i].planId].dailyInterest, investor.plans[i].investmentDate.add(investmentPlans_[investor.plans[i].planId].term), investor.plans[i].lastWithdrawalDate, investmentPlans_[investor.plans[i].planId].maxDailyInterest);
                    } else {
                        newDividends[i] = _calculateDividends(investor.plans[i].investment, investmentPlans_[investor.plans[i].planId].dailyInterest, block.timestamp, investor.plans[i].lastWithdrawalDate, investmentPlans_[investor.plans[i].planId].maxDailyInterest);
                    }
                } else {
                    newDividends[i] = _calculateDividends(investor.plans[i].investment, investmentPlans_[investor.plans[i].planId].dailyInterest, block.timestamp, investor.plans[i].lastWithdrawalDate, investmentPlans_[investor.plans[i].planId].maxDailyInterest);
                }
            }
        }
        return
        (
        investor.referrerEarnings,
        investor.availableReferrerEarnings,
        investor.referrer,
        investor.planCount,
        currentDividends,
        newDividends
        );
    }
```

The public view function getInvestorInfoByUID() will show various details of investors by their UID like referrerEarnings, availableReferrerEarnings, referrer, planCount, currentDividends. There are vulnerabilities or bugs here.

```
 function getInvestmentPlanByUID(uint256 _uid) public view returns (uint256[] m
emory, uint256[] memory, uint256[] memory, uint256[] memory, uint256[] memory,u
int256[] memory, bool[] memory) {
        if (msg.sender != owner) {
            require(address2UID[msg.sender] == _uid, "only owner or self can ch
eck the investment plan info.");
        }
        Objects.Investor storage investor = uid2Investor[_uid];
        uint256[] memory planIds = new  uint256[](investor.planCount);
        uint256[] memory investmentDates = new  uint256[](investor.planCount);
        uint256[] memory investments = new  uint256[](investor.planCount);
        uint256[] memory currentDividends = new  uint256[](investor.planCount);
        bool[] memory isExpireds = new  bool[](investor.planCount);
        uint256[] memory newDividends = new uint256[](investor.planCount);
        uint256[] memory interests = new uint256[](investor.planCount);

        for (uint256 i = 0; i < investor.planCount; i++) {
            require(investor.plans[i].investmentDate!=0,"wrong investment date"
);
            planIds[i] = investor.plans[i].planId;
            currentDividends[i] = investor.plans[i].currentDividends;
            investmentDates[i] = investor.plans[i].investmentDate;
            investments[i] = investor.plans[i].investment;
            if (investor.plans[i].isExpired) {
                isExpireds[i] = true;
                newDividends[i] = 0;
                interests[i] = investmentPlans_[investor.plans[i].planId].daily
Interest;
            } else {
                isExpireds[i] = false;
                if (investmentPlans_[investor.plans[i].planId].term > 0) {
                    if (block.timestamp >= investor.plans[i].investmentDate.add
(investmentPlans_[investor.plans[i].planId].term)) {
                        newDividends[i] = _calculateDividends(investor.plans[i]
.investment, investmentPlans_[investor.plans[i].planId].dailyInterest, investor
.plans[i].investmentDate.add(investmentPlans_[investor.plans[i].planId].term),
investor.plans[i].lastWithdrawalDate, investmentPlans_[investor.plans[i].planId
].maxDailyInterest);
                        isExpireds[i] = true;
                        interests[i] = investmentPlans_[investor.plans[i].planI
d].dailyInterest;
                    }
                    else{
                        newDividends[i] = _calculateDividends(investor.plans[i]
.investment, investmentPlans_[investor.plans[i].planId].dailyInterest, block.ti
mestamp, investor.plans[i].lastWithdrawalDate, investmentPlans_[investor.plans[
i].planId].maxDailyInterest);
                        interests[i] = investmentPlans_[investor.plans[i].planI
d].maxDailyInterest;
```

```
                    }
                } else {
                    newDividends[i] = _calculateDividends(investor.plans[i].inv
estment, investmentPlans_[investor.plans[i].planId].dailyInterest, block.timest
amp, investor.plans[i].lastWithdrawalDate, investmentPlans_[investor.plans[i].p
lanId].maxDailyInterest);
                    interests[i] =  investmentPlans_[investor.plans[i].planId].
maxDailyInterest;
                }
            }
        }

        return
        (
        planIds,
        investmentDates,
        investments,
        currentDividends,
        newDividends,
        interests,
        isExpireds
        );
    }
```

The public view function getInvestmentPlanByUID() will show various details of investmentplans by their UID like, planIDs, investmentDates, investments, currentDividends, newDividends, interests, isExpireds details.

```
    function _addInvestor(address _addr, uint256 _referrerCode) private returns
 (uint256) {
        if (_referrerCode >= REFERRER_CODE) {
            //require(uid2Investor[_referrerCode].addr != address(0), "Wrong re
ferrer code");
            if (uid2Investor[_referrerCode].addr == address(0)) {
                _referrerCode = 0;
            }
        } else {
            _referrerCode = 0;
        }
        address addr = _addr;
        latestReferrerCode = latestReferrerCode.add(1);
        address2UID[addr] = latestReferrerCode;
        emit Registration(addr,latestReferrerCode,uid2Investor[_referrerCode].a
ddr,_referrerCode);
        uid2Investor[latestReferrerCode].addr = addr;
        uid2Investor[latestReferrerCode].referrer = _referrerCode;
        uid2Investor[latestReferrerCode].planCount = 0;
        if (_referrerCode >= REFERRER_CODE) {

            uint256 _ref1 = _referrerCode;

            uid2Investor[_ref1].partners = uid2Investor[_ref1].partners.add(1);

        }
        return (latestReferrerCode);
    }
```

Private view function _addInvestor() is there to add new investor in the program with referrercode details and returning the latestReferrerCode.

```
    function _invest(address _addr, uint256 _planId, uint256 _referrerCode, uin
t256 _amount) private returns (bool) {
        require(_planId == 0, "Wrong investment plan id");
        require(_amount>= MINIMUM && _amount%700 trx==0, "Invalid Amount");
        uint256 uid = address2UID[_addr];
        if (uid == 0) {
            uid = _addInvestor(_addr, _referrerCode);
            //new user
        } else {//old user
            //do nothing, referrer is permenant
        }
        uint256 planCount = uid2Investor[uid].planCount;
        Objects.Investor storage investor = uid2Investor[uid];
        investor.plans[planCount].planId = _planId;
        investor.plans[planCount].investmentDate = block.timestamp;
        investor.plans[planCount].lastWithdrawalDate = block.timestamp;
        investor.plans[planCount].investment = _amount;
        investor.plans[planCount].currentDividends = 0;
        investor.plans[planCount].isExpired = false;

        investor.planCount = investor.planCount.add(1);
        Objects.Investor storage upline=uid2Investor[investor.referrer];
        for(uint256 i = 0; i < 10; i++) {
                if (upline.addr != address(0)) {
                        if(upline.partners>4)
                        {
                         address(uint160(upline.addr)).transfer((_amount*3)/100
);
                         emit UserIncome(uid2Investor[_referrerCode].addr, _add
r,i+1, 2,  (_amount*3)/100);
                        }
                    upline = uid2Investor[upline.referrer];
                } else break;
            }

        totalInvestments_ = totalInvestments_.add(_amount);
        uint256 directIncome=(_amount.mul(REFERENCE_RATE)).div(1000);
        address(uint160(uid2Investor[_referrerCode].addr)).transfer(directIncom
e);
        emit UserIncome(uid2Investor[_referrerCode].addr, _addr, 1, 1, directIn
come);


        uint256 rplanCount = uid2Investor[_referrerCode].planCount;
        Objects.Investor storage rinvestor = uid2Investor[_referrerCode];
        rinvestor.plans[rplanCount].planId = 1;
        rinvestor.plans[rplanCount].investmentDate = block.timestamp;
        rinvestor.plans[rplanCount].lastWithdrawalDate = block.timestamp;
        rinvestor.plans[rplanCount].investment = (_amount.mul(15)).div(1000);
```

```
        rinvestor.plans[rplanCount].currentDividends = 0;
        rinvestor.plans[rplanCount].isExpired = false;
        rinvestor.planCount = rinvestor.planCount.add(1);
        emit onInvest(rinvestor.addr, (_amount.mul(15)).div(1000),2);

        uint256 marketingPercentage = (_amount.mul(ADMIN_ENTRY_RATE)).div(1000)
;
        marketingAccount_.transfer(marketingPercentage);

        return true;
    }
```

Private view function _invest() is there for calculating investments different calculations and ultimately emit the UserIncome with details like, planID, investmentDate, lastWithdrawalDate, investment, currentDividends.

```
 function _ainvest(address _addr, uint256 _planId, uint256 _referrerCode, uint2
56 _amount) private returns (bool) {
        require(_planId == 0, "Wrong investment plan id");
        require(_amount>= MINIMUM && _amount%700 trx==0, "Invalid Amount");
        uint256 uid = address2UID[_addr];
        if (uid == 0) {
            uid = _addInvestor(_addr, _referrerCode);
            //new user
        } else {//old user
            //do nothing, referrer is permenant
        }
        uint256 planCount = uid2Investor[uid].planCount;
        Objects.Investor storage investor = uid2Investor[uid];
        investor.plans[planCount].planId = _planId;
        investor.plans[planCount].investmentDate = block.timestamp;
        investor.plans[planCount].lastWithdrawalDate = block.timestamp;
        investor.plans[planCount].investment = _amount;
        investor.plans[planCount].currentDividends = 0;
        investor.plans[planCount].isExpired = false;

        investor.planCount = investor.planCount.add(1);

        totalInvestments_ = totalInvestments_.add(_amount);


        uint256 rplanCount = uid2Investor[_referrerCode].planCount;
        Objects.Investor storage rinvestor = uid2Investor[_referrerCode];
        rinvestor.plans[rplanCount].planId = 1;
        rinvestor.plans[rplanCount].investmentDate = block.timestamp;
        rinvestor.plans[rplanCount].lastWithdrawalDate = block.timestamp;
        rinvestor.plans[rplanCount].investment = (_amount.mul(15)).div(1000);
        rinvestor.plans[rplanCount].currentDividends = 0;
        rinvestor.plans[rplanCount].isExpired = false;
        rinvestor.planCount = rinvestor.planCount.add(1);
        emit onInvest(rinvestor.addr, (_amount.mul(15)).div(1000),2);
        return true;
    }
```

Private view function _ainvest() is there for calculating different calculations for the added investments on the original investments.

```
    function invest(uint256 _referrerCode, uint256 _planId) public payable {
        if (_invest(msg.sender, _planId, _referrerCode, msg.value)) {
            emit onInvest(msg.sender, msg.value,1);
        }
    }

    function admininvest(address _user, uint256 _referrerCode, uint256 _planId,
 uint256 _amount) public payable {
         require(msg.sender == owner, "onlyOwner");
        if (_ainvest(_user, _planId, _referrerCode, _amount*1e6)) {
            emit onInvest(_user, _amount*1e6,1);
        }
    }

    function withdraw() public payable {
        require(msg.value == 0, "withdrawal doesn't allow to transfer trx simul
taneously");
        uint256 uid = address2UID[msg.sender];
        require(uid != 0, "Can not withdraw because no any investments");
        uint256 withdrawalAmount = 0;
        for (uint256 i = 0; i < uid2Investor[uid].planCount; i++)
        {
            if (uid2Investor[uid].plans[i].isExpired) {
                continue;
            }

            Objects.Plan storage plan = investmentPlans_[uid2Investor[uid].plan
s[i].planId];

            bool isExpired = false;
            uint256 withdrawalDate = block.timestamp;
            if (plan.term > 0) {
                uint256 endTime = uid2Investor[uid].plans[i].investmentDate.add
(plan.term);
                if (withdrawalDate >= endTime) {
                    withdrawalDate = endTime;
                    isExpired = true;
                }
            }

            uint256 amount = _calculateDividends(uid2Investor[uid].plans[i].inv
estment , plan.dailyInterest , withdrawalDate , uid2Investor[uid].plans[i].last
WithdrawalDate , plan.maxDailyInterest);

            withdrawalAmount += amount;


            uid2Investor[uid].plans[i].lastWithdrawalDate = withdrawalDate;
            uid2Investor[uid].plans[i].isExpired = isExpired;
```

```
            uid2Investor[uid].plans[i].currentDividends += amount;
        }



        uint256 marketingPercentage = (withdrawalAmount.mul(100)).div(1000);
        marketingAccount_.transfer(marketingPercentage);
        msg.sender.transfer((withdrawalAmount.mul(900)).div(1000));

        if (uid2Investor[uid].availableReferrerEarnings>0) {
            msg.sender.transfer(uid2Investor[uid].availableReferrerEarnings);
            uid2Investor[uid].referrerEarnings = uid2Investor[uid].availableRef
errerEarnings.add(uid2Investor[uid].referrerEarnings);
            uid2Investor[uid].availableReferrerEarnings = 0;
        }

        emit onWithdraw(msg.sender, withdrawalAmount);
    }
```

Public view payable functions invest(), admininvest(), withdraw() are there as a payable functions of investors funds and their withdrawing services.

```
    function _calculateDividends(uint256 _amount, uint256 _dailyInterestRate, u
int256 _now, uint256 _start , uint256 _maxDailyInterest) private pure returns (
uint256) {

        uint256 numberOfDays =  (_now - _start) / INTEREST_CYCLE ;
        uint256 result = 0;
        uint256 index = 0;
        if(numberOfDays > 0){
          uint256 secondsLeft = (_now - _start);
           for (index; index < numberOfDays; index++) {
               if(_dailyInterestRate + index <= _maxDailyInterest ){
                   secondsLeft -= INTEREST_CYCLE;
                     result += (_amount * (_dailyInterestRate + index) / 1000 *
 INTEREST_CYCLE) / (60*60*24);
               }
               else
               {
                 break;
               }
           }

            result += (_amount * (_dailyInterestRate + index) / 1000 * secondsL
eft) / (60*60*24);

            return result;

        }else{
            return (_amount * _dailyInterestRate / 1000 * (_now - _start)) / (6
0*60*24);
        }

    }
```

Private pure function _calculateDividends() is responsible for calculating the various dividends amount according to conditions for the investors.

Basic Coding Bugs

1. **Constructor Mismatch**
   - Description: Whether the contract name and its constructor are not identical to each other.
   - Result: PASSED
   - Severity: Critical
2. **Ownership Takeover**
   - Description: Whether the set owner function is not protected.
   - Result: PASSED
   - Severity: Critical
3. **Redundant Fallback Function**
   - Description: Whether the contract has a redundant fallback function.
   - Result: PASSED
   - Severity: Critical
4. **Overflows & Underflows**
   - Description: Whether the contract has general overflow or underflow vulnerabilities
   - Result: PASSED
   - Severity: Critical
5. **Reentrancy**
   - Description: Reentrancy is an issue when code can call back into your contract and change state, such as withdrawing ETHs.
   - Result: PASSED
   - Severity: Critical
6. **MONEY-Giving Bug**
   - Description: Whether the contract returns funds to an arbitrary address.
   - Result: PASSED
   - Severity: High

7. **Blackhole**

   - Description: Whether the contract locks ETH indefinitely: merely in without out.
   - Result: PASSED
   - Severity: High

8. **Unauthorized Self-Destruct**

   - Description: Whether the contract can be killed by any arbitrary address.
   - Result: PASSED
   - Severity: Medium

9. **Revert DoS**

   - Description: Whether the contract is vulnerable to DoS attack because of unexpected revert.
   - Result: PASSED
   - Severity: Medium

10. **Unchecked External Call**

    - Description: Whether the contract has any external call without checking the return value.
    - Result: PASSED
    - Severity: Medium

11. **Gasless Send**

    - Description: Whether the contract is vulnerable to gasless send.
    - Result: PASSED
    - Severity: Medium

12. **Send Instead of Transfer**

    - Description: Whether the contract uses send instead of transfer.
    - Result: PASSED
    - Severity: Medium

13. **Costly Loop**

- Description: Whether the contract has any costly loop which may lead to Out-Of-Gas exception.
- Result: PASSED
- Severity: Medium

14. **(Unsafe) Use of Untrusted Libraries**

- Description: Whether the contract use any suspicious libraries.
- Result: PASSED
- Severity: Medium

15. **(Unsafe) Use of Predictable Variables**

- Description: Whether the contract contains any randomness variable, but its value can be predicated.
- Result: PASSED
- Severity: Medium

16. **Transaction Ordering Dependence**

- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: PASSED
- Severity: Medium

17. **Deprecated Uses**

- Description: Whether the contract use the deprecated tx.origin to perform the authorization.
- Result: PASSED
- Severity: Medium

Semantic Consistency Checks

- Description: Whether the semantic of the white paper is different from the implementation of the contract.
- Result: PASSED
- Severity: Critical

Conclusion

In this audit, we thoroughly analyzed ETHRON' s Smart Contract. The current code base is well organized, well executed. The code was compiled and tested using compiler version 0.5.4 without errors. The same code is deployed and verified on main-net and currently also listed on dApp radar.

Meanwhile, we need to emphasize that smart contracts as a whole are still in an early, but exciting stage of development. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

About eNebula Solutions

We believe that people have a fundamental need to security and that the use of secure solutions enables every person to more freely use the Internet and every other connected technology. We aim to provide security consulting service to help others make their solutions more resistant to unauthorized access to data & inadvertent manipulation of the system. We support teams from the design phase through the production to launch and surely after.

The eNebula Solutions team has skills for reviewing code in C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities & specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. Additionally, the team can utilize various tools to scan code & networks and build custom tools as necessary.

Although we are a small team, we surely believe that we can have a momentous impact on the world by being translucent and open about the work we do.

Approved By: -
Director at
eNebula Solutions Pvt. Ltd.

Vatsal Raychura