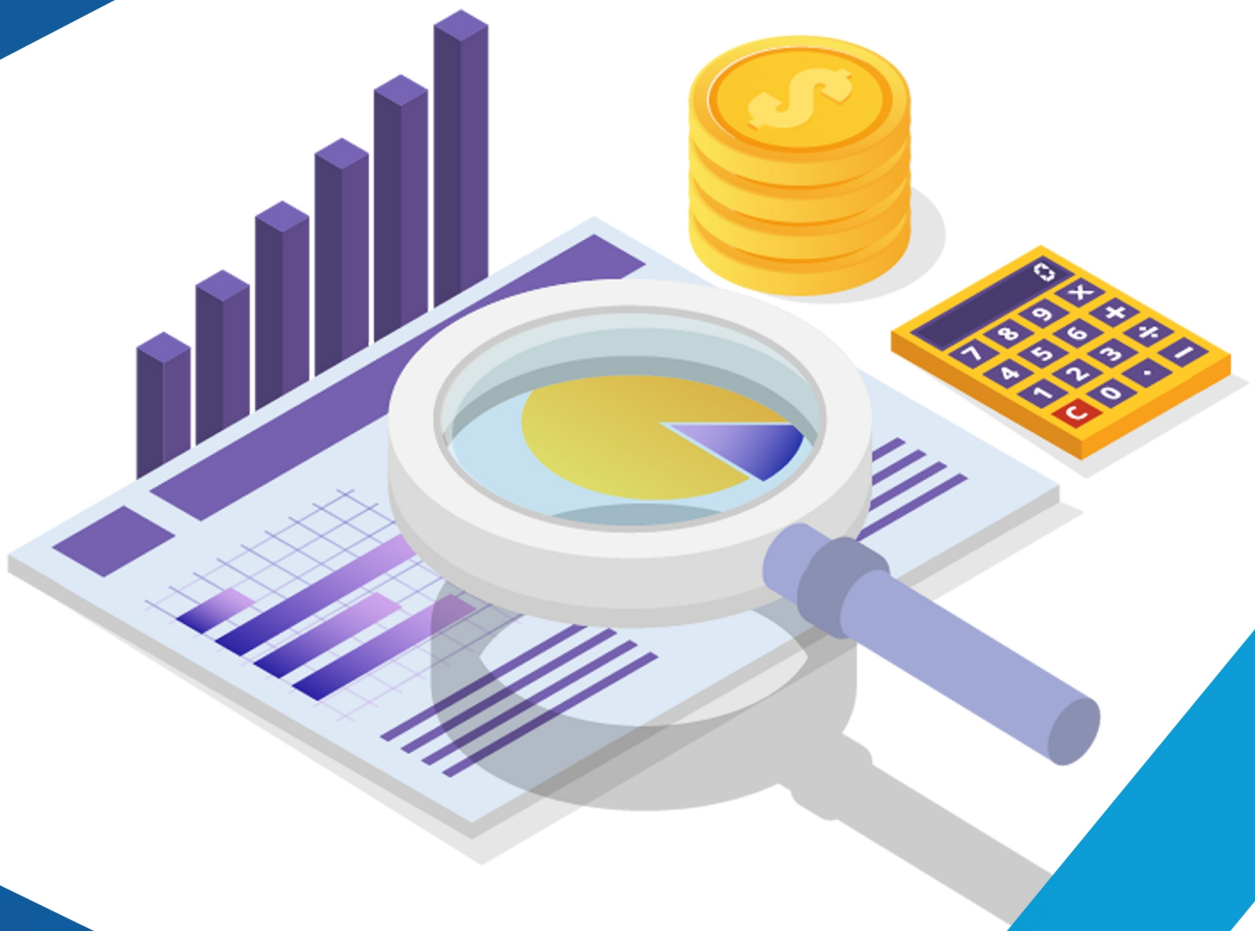
QuillAudits



Audit Report
December, 2020

UnionTron

Contents

Introduction

During the period of **December 21st, 2020 to December 23rd, 2020** - Quillhash Team performed a security audit for UnionTron smart contract. The code for audit was taken from following the official smart contract link:

Deployed at:
https://tronscan.org/#/contract/TBSM4fqJkKZ85tHD4UbSPixnVxsrK4A1kT/code

Description
The UnionTron Contract is fully based on the TRON blockchain and is a part of a community-based project. It simply allows participants to contribute TRX towards the community fund and receive support back from the community members themselves.

Imports
Null

Functions
- _setUpline(address _addr, address _upline)
- _deposit(address _addr, uint256 _amount)
- _pollDeposits(address _addr, uint256 _amount)
- refPayout(address _addr, uint256 _amount)
- drawPool()
- withdraw()
- payoutOf(address _addr)

Events
- event Upline(address indexed addr, address indexed upline);
- event NewDeposit(address indexed addr, uint256 amount);
- event DirectPayout(address indexed addr, address indexed from, uint256 amount);
- event MatchPayout(address indexed addr, address indexed from, uint256 amount);
- event PoolPayout(address indexed addr, uint256 amount);
- event Withdraw(address indexed addr, uint256 amount);
- event LimitReached(address indexed addr, uint256 amount);

Scope of Audit

The scope of this audit was to analyze and document UnionTron smart contract codebase for quality, security, and correctness.

Check Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per the intended behave mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

Static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Truffle Team, Ganache, Solhint, Mythril, Slither, SmartCheck.

Issue Categories

Every issue in this report has been assigned to a severity level. There are four levels of severity and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality and we recommend these issues to be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity four issues which indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

	Low	Medium	High	Informational
Open	5	2	0	3
Closed	0	0	0	0

Issues Found – Code Review / Manual Testing

Low Severity Issues

1. **External Visibility should be preferred**
   Description:
   The functions that are never called throughout the contract should be marked as **external** visibility instead of **public** visibility.
   This will effectively result in Gas Optimization as well.
   The **viewUserReferrals()** has been marked as public but is never called throughout the contract.

   Recommendation:
   Therefore, the following function must be marked as **external** within the contract:
   - viewUserReferrals() #Line 316

2. **Checking for ZERO Addresses**
   Description:
   UnionTron contract's constructor doesn't check for **Zero Addresses** before initializing some of the most imperative State Variables.

   Recommendation:
   The constructor should ensure a **Zero Address Validation.**

3. **Fallback function demands too much Gas**
   Description:
   The fallback function requires a larger amount of gas.
   Note:
   If the requirement of gas by the fallback function increases more than 2300, the Contract won't be able to receive Ether.

   Recommendation:
   Optimization of Gas must be considered.

4. **Use of "this" for Local Function:**

Description:

The contract uses the **this** keyword to call functions in the same contract at the following lines:

- **_deposit()** function at #Line127
- **withdraw()** function at #Line242
- **payoutOf()** function at #Line318

5. **Loops are extremely costly**

Description:

Every **loop** in the UnionTron contract include state variables like **.length** of a non-memory array, in the condition of the **for loops.**

As a result, these state variables consumes a lot more extra gas for every iteration of the **for** loop.

The following functions include such loops at the mentioned lines:

- **_pollDeposits** at #Line167, #Line176, #Line 178
- **_setUpline** at #Line 111
- **_drawPool** at #Line220, #Line231
- **_refPayout** at #Line199
- **poolTopInfo** at #Line345

Recommendation:

Its quite effective to use a **local variable** instead of **state variable** like **.length** in a loop.

For instance,

```
local_variable = ref_bonuses.length
for(uint8 i = 0; i < local_variable ; i++) {
        if(_upline == address(0)) break;
        users[_upline].total_structure++;
        _upline = users[_upline].upline;
 }
```

Medium Severity Issues

1. **Division Before Multiplication:**
   Description:
   The UnionTron contract performs multiplication on the result of division at some parts. This may result in the loss of precision. Solidity integer division might truncate.

   The following functions include multiplication before division at the mentioned lines:
   - **_drawPool()** at #Line218
   - **payoutOf()** at #Line321

   Recommendation:
   Multiplication operations should be performed before division.

3. **Withdraw function violates the Check-Effect-Interactions Pattern**
   Description:
   There is a potential violation of the **Check-Effect-Interactions** pattern in the **withdraw** function of the UnionTron Contract.
   Data is being read after the external call. This is not a recommended approach.

   Recommendation:
   Follow the **Check-Effect-Interactions Pattern.**

High Severity Issues

No high severity issues

Informational Issues

1. **UnionTron contract doesn't include the SafeMath Libraries**
   Since there are enormous arithmetic operations within the UnionTron Contract, it is recommended to use the SafeMath libraries from OpenZeppelin.

2. **Structs should be used for Multiple Return Value.**
   In order to improve the readability of the code, structs could be used instead of multiple return values.

3. **Coding Style Issues**
   Coding style issues influence code readability and in some cases may lead to bugs in future. Smart Contracts have naming convention, indentation and code layout issues. It's recommended to use the **Solidity Style Guide** to fix all the issues.
   Consider following the Solidity guidelines on formatting the code and commenting for all the files. It can improve the overall code quality and readability.

Closing Summary

1. Overall, the smart contracts are very well written and adhere to guidelines.

2. Two Medium severity issues, several issues of low severity and a few informational issues were found during the audit.

3. No instance of backdoor withdrawal of funds was found during the audit.

4. There were no critical or major issues found that can break the intended behavior

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the UnionTron platform. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough.
We recommend that the UnionTron Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

UnionTron

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

hello@quillhash.com