QuillAudits

Audit Report
May, 2021

Random Defi

Contents

Scope of Audit

The scope of this audit was to analyze and document Rand 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 behaviour.
- Token distribution and calculations are as per the intended behaviour 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.
SmartCheck.

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 security of smart contracts.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerability 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 automated analysis were manually verified.

Gas Consumption
In this step we have checked the behaviour 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 with 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 level 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 level 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

	High	Medium	Low	Informational
Open	0	0	1	2
Closed	1	0	4	0

Introduction

During the period of **MAY 6th, 2021 to May 8th, 2021** - Quillhash Team performed a security audit for Rand smart contracts.

The code for the audit was taken from following the official Github link:

https://testnet.bscscan.com/address/0x58aAbd4c75Ebd60f4e1B4E30A844C5105C336D7a#code

Issues Found – Code Review / Manual Testing

High severity issues

1. Contract completely locks Ether and fails to provide a way to Unlock it
   Line no: 465
   Status: CLOSED.

   Description:
   The constructor of the contract includes the **payable** keyword. It indicates that the contract allows the transfer of **ETHER** into the contract.

   However, no function to withdraw Ether from the contract was found.

   This could lead to a very undesirable situation where any Ether sent to the contract will be completely lost and unrecoverable.

```
464     */
465     constructor(address payable _tokenOwnerAddress) public payable {
466
```

   Moreover, a similar warning was found while conducting the automated testing of the contract.

```
Contract locking ether found in :
        Contract RandToken (contracts/RAND.sol#444-556) has payable functions:
         - RandToken.constructor(address) (contracts/RAND.sol#465-474)
        But does not have a function to withdraw the ether
```

Recommendation:
The contract should either remove the **payable keyword** from the **constructor** function or include a function that allows the withdrawal of the locked ETHER in the contract.

Note:
If the contract is not supposed to recieve ether, the **payable** keyword should be removed from the constructor as well. (Because the contract can still accept ether via constructor, while deployment).
Otherwise, a mechanism to withdraw the locked ether in the contract should be implemented.

Low level severity issues

1. **Absence of Error messages in Require Statements**

Line no - 668

Status: CLOSED

Description:
The **require** statements in the Rand.sol contract that do not include any error message.

While this makes it troublesome to detect the reason behind a particular function revert, it also reduces the readability of the code.

Recommendation:
Error Messages must be included in every required statement in the contract.

2. **External Visibility should be preferred**

Status: CLOSED

Explanation:
Those 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.

Therefore, the following function must be marked as **external** within the contract:

- **initialSupply**
- **emitEveryWeekTokens**
- **setGuessAddress**

Recommendation:
If the PUBLIC visibility of the above-mentioned functions is not intended, then the EXTERNAL Visibility keyword should be preferred.

3. **SafeMath Library should be used to perform arithmetic operations**
   Line : 482

   Status: Not Considered

Description:
The transfer function includes a IF_ELSE condition at the very beginning of the function to ensure whether or not the elapsed time is greater than 72 hours.

```
481     function transfer(address recipient, uint256 _amount)
482         if ((now-contractDeploymentTime) > 72 hours ) {
483             uint fee;
```

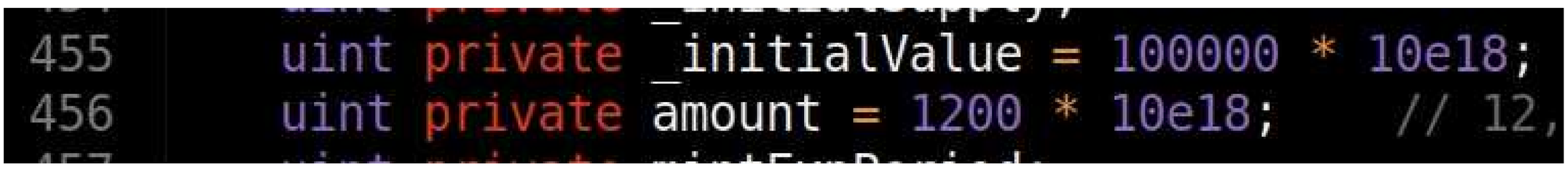
However, the subtraction operation is performed using the “-” operator instead of .sub() function of Safemath library.

Recommendation:
It is considered a better practice to use SafeMath library for performing arithmetic operations in Solidity.

4. **Too many Digits used**
   Line no - 459, 460, 647

   Status: CLOSED

Description:
The above-mentioned lines have a large number of digits that makes it difficult to review and reduces the readability of the code.



Recommendation:
Ether Suffix should be used to symbolize the 10^18 zeros.
Note:
During the final Audit review, a similar issue was found in one of the local variables of emitEveryWeekTokens function as well

5. **Contract includes Hardcoded Addresses**
   Line no - 458

Status: CLOSED

Description:
Keeping in mind the immutable nature of smart contracts, it is not considered a better practise to hardcode any address in the contract before deployment.
Most importantly, when that particular address is involved in token transfers.

Recommendation:
Instead of including hardcoded addresses in the contract, initialize those addresses within the constructors at the time of deployment.

Informational

1. Coding Style Issues in the Contract

Status: Not Considered

Explanation:
Code readability of a Smart Contract is largely influenced by the Coding Style issues and in some specific scenarios may lead to bugs in the future.

During the automated testing, it was found that the Rand contract had quite a few code style issues.

```
Token.transfer(address,uint256)._amount (contracts/RAND.sol#481) is not in m
Token.setGuessAddress(address)._guessAddress (contracts/RAND.sol#494) is not
Token.transferGuess(address,uint256)._amount (contracts/RAND.sol#502) is not
ns://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-so
```

Recommendation:
Therefore, it is highly recommended to fix the issues like naming convention, indentation, and code layout issues in a smart contract.

2. Coding Style Issues in the Contract

Status: Not Considered

Description:
The smart contracts do not include the NatSpec annotations adequately.

Recommendation:
Cover by NatSpec all Contract methods.

Closing Summary

Overall, smart contracts are very well written and adhere to guidelines. No instances of Re-entrancy or Back-Door Entry were found in the contract.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the **Rand 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 Rand Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Random Defi

QuillAudits

Canada, India, Singapore and Uniked Kingdom

audits.quillhash.com

audits@quillhash.com