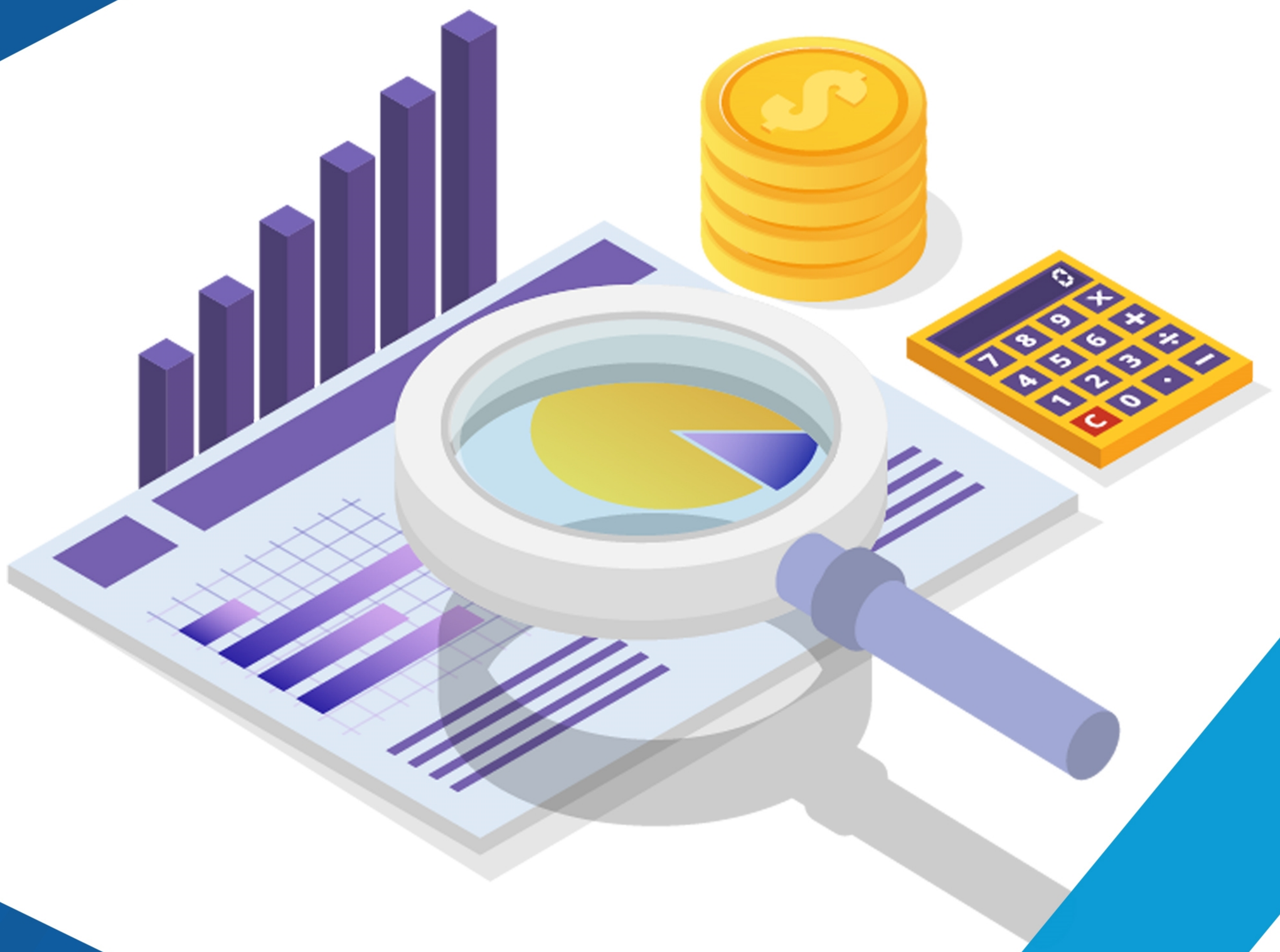
QuillAudits



Audit Report
March, 2021

Contents

Scope of Audit

The scope of this audit was to analyze and document Trees Finances 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	0	9
Closed	0	0	0	0

Introduction

During the period of **March 9th 2021 to March 12th 2021** - Quillhash Team performed a security audit for **Trees Finances** smart contracts.
The code for the audit was taken from following the official GitHub link:

https://github.com/treesfinance/Ganja/blob/main/treesFinance.sol

Commit Hash - 98e05c0680ac09b1efbe71eb5bdbdbbc1f51522c

Issues Found – Code Review / Manual Testing

A BEP-20 token contract with mintable and burnable features.

The contract includes an additional functionality of deducting a certain percentage(basePercent) of the token amount to be transferred and transferring it to a predefined address.

High severity issues

No Issues found.

Medium severity issues

No Issues found.

Low level severity issues

No Issues found.

Informational

1. **Excessive Gas Consumption in Loops**
   Line no - 145

   Description:
   The loops implemented in the above-mentioned line use the **.length** state variable of the respective array to dynamically decide the upper bound of the iteration.
   However, for every iteration of for loop, state variables like .length of the non-memory array will consume comparatively more gas.
   Therefore, it would be more effective to use a local variable instead of a state variable like **.length** in a loop

   Recommendation:
   In order to reduce the extra gas consumption, it's advisable to use a local variable.
   The example attached below can be taken as a reference.

```
146 ▾  function multiTransfer(address[] memory receivers, uint256[] memory amounts) public {
147      uint256 localVariable = receivers.length;
148      for (uint256 i = 0; i < localVariable; i++) {
149        transfer(receivers[i], amounts[i]);
150      }
151    }
```

2. **External visibility should be preferred**

   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:

   - totalSupply()
   - balanceOf(address)
   - allowance(address,address)
   - multiTransfer(address[],uint256[])
   - approve(address,uint256)
   - transferFrom(address,address,uint256)
   - increaseAllowance(address,uint256)
   - decreaseAllowance(address,uint256)

Recommendation:
The above-mentioned functions should be assigned external visibility.

3. **Internal visibility should be preferred**
   Line no - 119

   Description:
   Functions that are to be called only by the contract itself should be marked with **internal** visibility instead of **public** visibility.
   This will make the code more readable and is considered a better practice.

   Therefore, the following function must be marked as **internal** within the contract:
   - findPercent(uint256 value)

   Recommendation:
   The above-mentioned functions should be assigned internal visibility.

4. **Excessive Gas Consumption in Loops**
   Line no - 145

   Description:
   None of the **require** statements in the contract includes an 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 should be included in every require statement.

```
146 ▾  function multiTransfer(address[] memory receivers, uint256[] memory amounts) public {
147      uint256 localVariable = receivers.length;
148      for (uint256 i = 0; i < localVariable; i++) {
149        transfer(receivers[i], amounts[i]);
150      }
151    }
```

5. **Ceil function is never used throughout the Contract**
   Line no - 57

   Description:
   The Safemath library in the contract includes an additional function called **Ceil**.
   However, the function is never used throughout the contract.

```
57    function ceil(uint256 a, uint256 m) internal pure returns (uint256) {
58      uint256 c = add(a,m);
59      uint256 d = sub(c,1);
60      return mul(div(d,m),m);
61    }
62  }
```

   Recommendation:
   Eliminate all unwanted and unused functions/variables.

6. **Constant declaration should be preferred**
   Line no- 100

   Description:
   State variables that are not supposed to change throughout the contract should be declared as constant.

   Recommendation:
   The following state variables need to be declared as constant:
   - basePercent

7. **Explicit visibility declaration is missing**
   Line no- 97, 98,99, 100

   Description:
   The state variables in the above-mentioned lines have not been assigned any visibility explicitly.

   Recommendation:
   Visibility specifiers should be assigned explicitly in order to avoid ambiguity.

8. **Too many Digits used**
   Line no - 100

   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.

9. **Contract includes Hardcoded Addresses**
   Line no- 134 & 167

   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.

Closing Summary

Overall, smart contracts are well written and adhere to guidelines.
During the final audit procedure, no high, medium were found.
No instances of Re-entrancy or Backdoor Entry were found as well.
Some imperative gas usage issues, however, have been addressed. These shall not effect the intended execution of the contract but are only reported for gas optimization purposes.

Disclaimer

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

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

hello@quillhash.com