# Trees Finances Smart Contract Initial Audit Report

## 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.

## 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 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 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

| TYPE | HIGH | MEDIUM | LOW | INFORMATIONAL |
|---|---|---|---|---|
| OPEN | 4 | 3 | 5 | 2 |
| 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 - 95f8dc91056b07ec0047d71a96eeccc5a0199dff***

## A. Contract Details
## Name - TreesFinance

**About the contract:**
An 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.

# Issues Found – Code Review / Manual Testing

## High Severity Issues

### A.1 Mint Function Never Increases the Total Supply of the Token

**Line no - 197 - 201**

***Description:***

The token includes an internal **_mint** function that doesn't increase the total supply of the token when called by the admin.

This will lead to an unwanted scenario where keeping track of the total tokens in circulation will be troublesome.

```
197     function _mint(address account, uint256 amount) internal {
198       require(amount != 0);
199       _balances[account] = _balances[account].add(amount);
200       emit Transfer(address(0), account, amount);
201     }
```

***Recommendation:***

Follow the Token Standards procedure for minting new tokens and update the mint function accordingly.

### A.2 Total Supply of the token doesn't match with the Actual Supply

**Line no - 99 and 105**

***Description:***

The ***_totalSupply*** state variable of the token contract has initially been assigned a value of **5000 * 10^18.**

However, the actual tokens minted and transferred to the admin at the time of deployment is only **3000 * 10^18.** This results in a situation where the

actual amount of tokens in circulation is less than expected.

```
103    constructor() public payable ERC20Detailed(tokenName, tokenSymbol, tokenDecimals) {
104      admin = msg.sender;
105      mint(msg.sender, 300000000000000000000);
106    }
```

***Recommendation:***

The correct quantity of tokens should be minted for the admin at the time of contract deployment. The total supply should match the actual amount of tokens in circulation.

### A.3 Contract completely locks Ether and fails to provide a way to Unlock it

**Line no - 103**

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

```
103    constructor() public payable ERC20Detailed(tokenName, tokenSymbol, tokenDecimals) {
104      admin = msg.sender;
105      mint(msg.sender, 300000000000000000000);
106    }
```

***Recommendation:***

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

### A.4 Token Transfer tax is applicable even for the Admin/Owner Address

**Line no - 126 & 157**

***Description:***

The transfer and transferFrom function include an additional functionality of deducting a particular amount from the tokens being transferred.

However, this deduction of tokens is also applicable when the tokens are either being transferred ***to the owner*** or ***by the owner*** to a different address.

## Is this Intended?

```
129
130     uint256 tokensToBurn = findPercent(value);
131     uint256 tokensToTransfer = value.sub(tokensToBurn);
132
133     _balances[msg.sender] = _balances[msg.sender].sub(value);
134     _balances[to] = _balances[to].add(tokensToTransfer);
135     _balances[0x98B58134671b9219B461dD02191585F65753972e] = _balances[0x98B58134671b9219B461dD02191585F65753972e].add(tokensToBurn);
136
```

### *Recommendation:*

If this is not an intended functionality, the tokens transfer functions in the contract must be updated in a way that tokens aren't being deducted when the admin/owner is the sender or receiver of the tokens.

# Medium Severity Issues

### A.5 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

**Recommended way :**

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   }
```

### A.6 Tokens aren't being Burnt during transfer

### Line no - 130 & 164

***Description:***

The transfer and transferFrom function include a local variable called ***tokensToBurn*** to store the number of tokens to be deducted from the total tokens to be transferred.

However, this amount of tokens is never burnt but simply transferred to a hardcoded address. Moreover, since the tokens aren't being burnt, there is no change in the total supply as well.

```
129
130     uint256 tokensToBurn = findPercent(value);
131     uint256 tokensToTransfer = value.sub(tokensToBurn);
132
```

***Recommendation:***

If the tokens aren't supposed to be burnt during the transfers, the name of the local variable, i.e., ***tokensToBurn,*** must be changed to avoid any confusions within the community.

### A.7 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.

## Low Severity Issues

### A.8 External visibility should be preferred

***Description:***
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()  at Line  108-110*
- *balanceOf(address) at Line  112-114*
- *allowance(address,address) at Line 116-118*
- *multiTransfer(address[],uint256[]) at Line144-148*
- *approve(address,uint256) at Line 150-155*
- *transferFrom(address,address,uint256) at Line 157-176*
- *increaseAllowance(address,uint256) at Line 178-183*
- *decreaseAllowance(address,uint256) at Line 185-190*

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

## A.9 Internal visibility should be preferred

***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)* at Line 120

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

## A.10 Absence of Error messages in Require Statements

***Description:***
None of the ***require*** statements in the contract include 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.

## A.11 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***

### A.12 Explicit visibility declaration is missing

**Line no - 96, 97, 98,99, 101**

***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.

## Informational

### A.13 Too many Digits used

**Line no - 99 & 105**

***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.

### A.14 Contract includes Hardcoded Addresses

**Line no - 135 & 168**

***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.

However, during the process of audit, several issues of high, medium as well as low severity were found which might affect the intended behaviour of the contracts.

Therefore, it is recommended to check the above-mentioned issues and fix them to avoid any unexpected scenario during the execution of the contract.

## 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.