YearnyFi Network Smart Contract 1st Audit Report

Scope of Audit

The scope of this audit was to analyze and document YearnyFi.Network 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	0	1	3	2
CLOSED	0	0	0	0

Introduction

During the period of **February 8th, 2021 to February 10th, 2021** - Quillhash Team performed a security audit for **YearnyFi.Network** smart contract.

The deployed code for the audit can be found on the Etherscan link below**:** https://etherscan.io/address/0xdeaa665ab71785ccf576dc76e5fcb5a2283ea0c6#code

A. Contract Details
Name - YEARNYFINETWORK

About the Contract:
YEARNYFINETWORK is an ERC20 token contract that implements the basic functionalities of an ERC20 token standard.

The contract includes some of the basic functionalities of an ERC20 token like *transfer, approve, transferFrom* etc.

Issues Found – Code Review / Manual Testing

High Severity Issues

Not Found

Medium Severity Issues

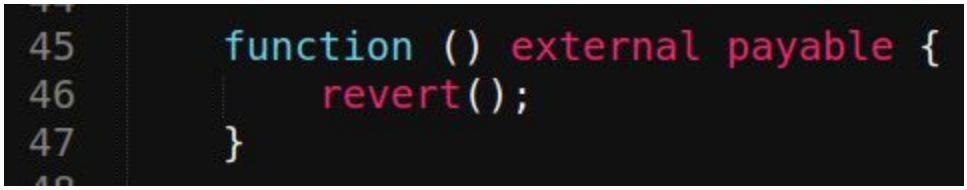
A.2 The inclusion of Fallback function is Redundant

Description:

The token contract includes a fallback function which simply reverts whenever it is triggered.

This, however, makes the payment rejection process inadequate. This is because, after Solidity version 0.4.0, a contract without fallback function will automatically revert the payment.

Hence, this doesn't require an additional fallback function within the code.



Recommendation:

If the fallback function is only supposed **revert,** it should be removed.

Low Severity Issues

A.3 Arithmetic Operations might lead to Integer Overflow or Underflow scenario

Description:
The token contract doesn't use the SafeMath library and implements the arithmetic operations simply.
Keeping in mind the way integers work in Solidity, this could lead to an Arithmetic Overflow or Underflow exploit scenario.

Recommendation:
Use SafeMath for all arithmetic operations.

A.4 Redundant Require Statements

Description:
The **transferFrom function** in the token contract includes redundant **require statements** to check the **sender(_*from*)** as well as the **receiver(_*to*) addresses.**

The **allowed function** in the require statements(*at Line 118 below*) already checks the allowance of the **caller of the function w.r.t to the owner of the tokens.**

Moreover, including multiple **require** statements for a similar logic will ultimately increase the Gas Usage.

```
111     function transferFrom(address _from, address _to, uint256 _value) public returns (bool) {
112         require(_from != address(0));
113         require(_from != address(this));
114         require(_to != _from);
115         require(_to != address(0));
116         require(_to != address(this));
117         require(_value <= transferableTokens(_from));
118         require(allowed[_from][msg.sender] - _value <= allowed[_from][msg.sender]);
119         require(balances[_from] - _value <= balances[_from]);
120         require(balances[_to] <= balances[_to] + _value);
121
```

Recommendation:
Remove the unnecessary require statements.

A.5 External visibility should be preferred

Description:
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.

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

- *allowance*
- *approve*
- *decreaseApproval*
- *increaseApproval*
- *transfer*
- *transferFrom*
- *transferOwnership*

Informational

A.6 Absence of Error messages in Require Statements

Description:

No require statement 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:

Include error messages in every require statement.

A.7 Compiler Version is not Fixed

Description:

The compiler version in the contract file is not fixed.
It is considered an effective practice to specify an exact compiler version.

Recommendation:

Compiler version should be fixed.

Closing Summary

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

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

It is recommended to fix the issues found during the audit in order to achieve an adequate execution of the smart contract.

Disclaimer

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