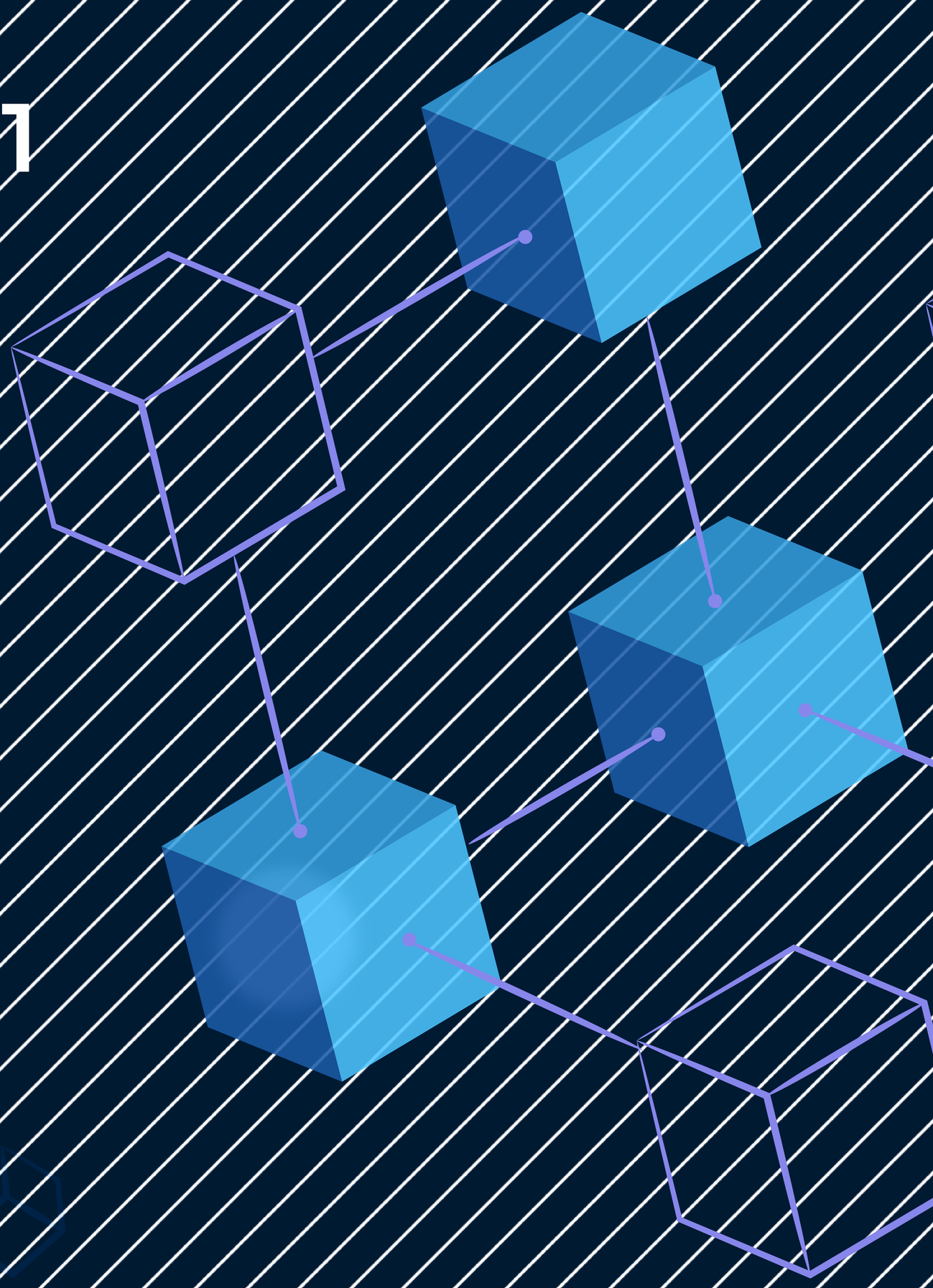
QuillAudits

Audit Report
December, 2021



For

Yearnlab

Contents

Scope of Audit	01
Check Vulnerabilities	01
Techniques and Methods	02
Issue Categories	03
Number of security issues per severity.	03
Introduction	04
Issues Found – Code Review / Manual Testing	05
High Severity Issues	05
Medium Severity Issues	05
Low Severity Issues	05
A.1 ERC20 approve() race-condition	05
A.2 Missing zero address validation	06
Informational Issues	06
A.3 Missing Events for Significant Transactions	06
A.4 Unused events	07
Functional Test Cases	08
Automated Tests	09
Results:	10
Closing Summary	11

Scope of the Audit

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

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- 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
- BEP20 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 contract, 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 BEP-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 analysed 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 analysed, 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 the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis, Theo.

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.

Risk-level	Description
High	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 be fixed before moving to a live environment.
Medium	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	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 issues that 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	0	0	0
Acknowledged	0	0	2	2
Closed	0	0	0	0

Introduction

During the period of **November 28, 2021 to December 1, 2021** - QuillAudits Team performed a security audit for **Yearnlab** smart contracts.

The code for the audit was taken from following the official link: https://github.com/Yearn-Lab/Yearnlab-Contract/blob/main/YearnlabToken

Ver	Date	Commit hash	Files
1	28 November	02ecc59290835bb5eea6 c6a8e4b60f3b1cbca89e	1. YearnlabToken

Issues Found

A. Contract – Yearnlab

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

1. **ERC20 approve() race-condition**

   Description
   Using approve() call inside the **approveAndCall** function to manage allowances opens yourself and users of the token up to front-running. Changing an allowance with this method brings the risk that someone may use both the old and the new allowance by unfortunate transaction ordering.
   Read more

   Remediation
   Implement the Openzeppelin's ERC20 increaseAllowance and decreaseAllowance functions.

   Status: Acknowledged

2. Missing zero address validation

Line	Code

182-184	function transferOwnership(address newOwner) public onlyOwner {
	owner = newOwner;
	}

Description
When setting the new owner address, it should be checked for **zero address**. Otherwise, they may lose the ability to use the privileged functions.

Remediation
Use the **require** statement to check for zero addresses.

Status: Acknowledged

Informational issues

3. Missing Events for Significant Transactions

Description
The missing event makes it difficult to track off-chain decimal changes. An event should be emitted for significant transactions calling the functions :
- transferOwnership

Remediation
We recommend emitting the appropriate events.

Status: Acknowledged

4. Unused events

Line	Code
342	event LogInt(string, uint256);

Description

There is an unused event in the codebase.

Remediation

Use the event in the appropriate function or remove it.

Status: Acknowledged

Functional Test Cases

- Should be able to transfer/transferFrom **PASS**
- totalSupply should equal the specifiedamount (100000000 * 10^18) **PASS**
- Increase and decrease allowance should update allowance **PASS**
- approveAndCall should update allowance and receiveApproval **PASS**

Automated Tests

Slither

```
Owned.transferOwnership(address) (Yearnlab.sol#182-184) should emit an event for:
        - owner = newOwner (Yearnlab.sol#183)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-con
trol

Owned.transferOwnership(address).newOwner (Yearnlab.sol#182) lacks a zero-check on :
                - owner = newOwner (Yearnlab.sol#183)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-vali
dation

Different versions of Solidity is used:
        - Version used: ['0.7.5', '>=0.4.22<0.9.0']
        - 0.7.5 (Yearnlab.sol#1)
        - >=0.4.22<0.9.0 (Migrations.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directiv
es-are-used

SafeMath.div(uint256,uint256) (Yearnlab.sol#102-104) is never used and should be removed
SafeMath.div(uint256,uint256,string) (Yearnlab.sol#119-130) is never used and should be removed
SafeMath.mod(uint256,uint256) (Yearnlab.sol#143-145) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (Yearnlab.sol#160-167) is never used and should be removed
SafeMath.mul(uint256,uint256) (Yearnlab.sol#77-89) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Yearnlab.DEC (Yearnlab.sol#245) is set pre-construction with a non-constant function or state vari
able:
        - 10 ** uint256(decimals)
Yearnlab._totalSupply (Yearnlab.sol#246) is set pre-construction with a non-constant function or s
tate variable:
        - 100000000 * DEC
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#function-initializing-sta
te
```

```
Variable Yearnlab.DEC (Yearnlab.sol#245) is not in mixedCase
Variable Yearnlab._totalSupply (Yearnlab.sol#246) is not in mixedCase
Variable Migrations.last_completed_migration (Migrations.sol#6) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-n
aming-conventions

Yearnlab.slitherConstructorVariables() (Yearnlab.sol#239-344) uses literals with too many digits:
        - _totalSupply = 100000000 * DEC (Yearnlab.sol#246)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

Yearnlab.decimals (Yearnlab.sol#244) should be constant
Yearnlab.name (Yearnlab.sol#243) should be constant
Yearnlab.symbol (Yearnlab.sol#242) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-coul
d-be-declared-constant

transferOwnership(address) should be declared external:
        - Owned.transferOwnership(address) (Yearnlab.sol#182-184)
totalSupply() should be declared external:
        - Yearnlab.totalSupply() (Yearnlab.sol#256-258)
balanceOf(address) should be declared external:
        - Yearnlab.balanceOf(address) (Yearnlab.sol#260-267)
transfer(address,uint256) should be declared external:
        - Yearnlab.transfer(address,uint256) (Yearnlab.sol#269-278)
increaseAllowance(address,uint256) should be declared external:
        - Yearnlab.increaseAllowance(address,uint256) (Yearnlab.sol#290-296)
decreaseAllowance(address,uint256) should be declared external:
        - Yearnlab.decreaseAllowance(address,uint256) (Yearnlab.sol#298-304)
transferFrom(address,address,uint256) should be declared external:
        - Yearnlab.transferFrom(address,address,uint256) (Yearnlab.sol#306-316)
allowance(address,address) should be declared external:
        - Yearnlab.allowance(address,address) (Yearnlab.sol#318-325)
approveAndCall(address,uint256,bytes) should be declared external:
        - Yearnlab.approveAndCall(address,uint256,bytes) (Yearnlab.sol#327-340)
setCompleted(uint256) should be declared external:
        - Migrations.setCompleted(uint256) (Migrations.sol#16-18)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
. analyzed (6 contracts with 75 detectors), 28 result(s) found
```

Results

No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

Closing Summary

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

No instances of Integer Overflow and Underflow vulnerabilities or Back-Door Entry were found in the contract, but relying on other contracts might cause Reentrancy Vulnerability.

Two Low Severity Issues were found during the Audit, which has been Acknowledged by the Yearnlab team.

Disclaimer

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

Audit Report

December, 2021

For

Yearnlab

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com