QuillAudits

Audit Report
April, 2021

Contents

Introduction

This audit report highlights the overall security of the APRIL-token contract. With this report, we have tried to ensure the reliability of the smart contract by completing the assessment of their system's architecture and smart contract codebase.

Auditing Approach and Methodologies applied

In this audit, I consider the following crucial features of the code.

- Whether the implementation of token standards.
- Whether the code is secure.
- Whether the code meets the best coding practices.
- Whether the code meets the SWC Registry issue.

The audit has been performed according to the following procedure:

Manual Audit

- Inspecting the code line by line and revert the initial algorithms of the protocol and then compare them with the specification
- Manually analyzing the code for security vulnerabilities.
- Gas Consumption and optimisation
- Assessing the overall project structure, complexity & quality.
- Checking SWC Registry issues in the code.
- Unit testing by writing custom unit testing for each function.
- Checking whether all the libraries used in the code of the latest version.
- Analysis of security on-chain data.
- Analysis of the failure preparations to check how the smart contract performs in case of bugs and vulnerability.

Automated analysis

- Scanning the project's code base with Mythril, Slither, Manticore, others.
- Manually verifying (reject or confirm) all the issues found by tools.
- Performing Unit testing.
- Manual Security Testing (SWC-Registry, Overflow)
- Running the tests and checking their coverage.

Audit Details

Project Name: April Token
Token symbol: APRIL

Languages: Solidity
Platforms and Tools: HardHat, Remix, VScode, solhint and other tools mentioned in the automated analysis section.

Audit Goals

The focus of this audit was to verify whether the smart contract is secure, resilient, and working according to ERC20 specs. The audit activity can be grouped into three categories.

Security
Identifying security related issues within each contract and the system of contract.

Sound Architecture
Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

Code Correctness and Quality
A full review of the contract source code. The primary areas of focus include:

- Correctness.
- Section of code with high complexity.
- Readability.
- Quantity and quality of test coverage.

Issue Categories

Every issue in this report was assigned a severity level from the following:

High severity issues

Issues on this level are critical to the smart contract's performance/ functionality and should be fixed before moving to a live environment.

Medium severity issues

Issues on this level could potentially bring problems and should eventually be fixed.

Low severity issues

Issues on this level are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Number of issues per severity

Severity	High	Medium	Low
Open	0	0	3

Manual Audit

SWC Registry test

We have tested SWC registry issues as well. Out of all tests, only SWC 100,104 and 114. All three found issues are low priority. We have put the details about it below in the low priority section.

Serial No.	Description	Comments
SWC-132	Unexpected Ether balance	Pass: Avoided strict equality checks for the Ether balance in a contract
SWC-131	Presence of unused variables	Pass: No unused variables
SWC-128	DoS With Block Gas Limit	Pass
SWC-122	Lack of Proper Signature Verification	Pass
SWC-120	Weak Sources of Randomness from Chain Attributes	Pass
SWC-119	Shadowing State Variables	Pass: No ambiguous found.
SWC-118	Incorrect Constructor Name	Pass. No incorrect constructor name used
SWC-116	Timestamp Dependence	Pass
SWC-115	Authorization through tx.origin	Pass: No tx.origin found
SWC-114	Transaction Order Dependence	Found

Serial No.	Description	Comments
SWC-113	DoS with Failed Call	Pass: No failed call
SWC-112	Delegatecall to Untrusted Callee	Pass
SWC-111	Use of Deprecated Solidity Functions	Pass : No deprecated function used
SWC-108	State Variable Default Visibility	Pass: Explicitly defined visibility for all state variables
SWC-107	Reentrancy	Pass: Properly used
SWC-106	Unprotected SELF-DESTRUCT Instruction	Pass: Not found any such vulnerability
SWC-104	Unchecked Call Return Value	Pass: Not found any such vulnerability
SWC-103	Floating Pragma	Pass
SWC-102	Outdated Compiler Version	Fixed: Latest version is Version 0.8.3. In code 0.5.16 is used
SWC-101	Integer Overflow and Underflow	Pass

High level severity issues

No issues found

Medium level severity issues

No issues found

Low level severity issues

1. **Description → SWC 102**: Outdated Compiler Version [Line no. 5]

```
2    *Submitted for verification at BscScan.com on 2021-04-24
3   */
4
5   pragma solidity 0.5.16;
6
7   interface IBEP20 {
8     /**
9      * @dev Returns the amount of tokens in existence.
```

Using an outdated compiler version can be problematic, especially if there are publicly disclosed bugs and issues that affect the current compiler version.

Remediation
It is recommended to use a recent version of the Solidity compiler, which is Version 0.8.3.

Status: Open

2. **Description → Using the approve function of the token standard [Line 434-437]**

```
431      *
432      * - `spender` cannot be the zero address.
433      */
434    function approve(address spender, uint256 amount) external returns (bool) {
435      _approve(_msgSender(), spender, amount);
436      return true;
437    }
438
```

The **approve** function of ERC-20 is vulnerable. Using a front-running attack, one can spend approved tokens before the change of allowance value.

To prevent attack vectors described above, clients should make sure to create user interfaces in such a way that they set the allowance first to 0 before setting it to another value for the same spender. However, the contract itself shouldn't enforce it to allow backward compatibility with contracts deployed before.
Detailed reading around it can be found at EIP 20

Status: Open

3. **Description: Prefer external to public visibility level [line 320-323, 329-331, 469-472, 488-490]**

```
319    */
320  function renounceOwnership() public onlyOwner {
321    emit OwnershipTransferred(_owner, address(0));
322    _owner = address(0);
323  }
324
325  /**
326   * @dev Transfers ownership of the contract to a new account (`newOwner`).
327   * Can only be called by the current owner.
328   */
329  function transferOwnership(address newOwner) public onlyOwner {
330    _transferOwnership(newOwner);
331  }
332
```

A function with a **public** visibility modifier that is not called internally. Changing the visibility level to **external** increases code readability. Moreover, in many cases, functions with **external** visibility modifiers spend less gas compared to functions with **public** visibility modifiers.

The function definition of **renounceOwnership(), transferOwnership(), increaseAllowance() and decreaseAllowance()** is marked as "public". However, it is never directly called by another function in the same contract or any of its descendants. Consider marking it as"external" instead.

Recommendations: Use the external visibility modifier for functions never called from the contract via internal call. Reading Link.

Note: Exact same issue was found while using automated testing by Mythx and Slither.

Status: Open

Automated Testing

We have used multiple automated testing frameworks. This makes code more secure common attacks. The results are below.

Slither

Slither is a Solidity static analysis framework that runs a suite of vulnerability detectors, prints visual information about contract details, and provides an API to easily write custom analyses. Slither enables developers to find vulnerabilities, enhance their code comprehension, and quickly prototype custom analyses. After running Slither, we got the results below.

```
INFO:Detectors:
BEP20Token.allowance(address,address).owner (april.sol#423) shadows:
        - Ownable.owner() (april.sol#301-303) (function)
BEP20Token._approve(address,address,uint256).owner (april.sol#528) shadows:
        - Ownable.owner() (april.sol#301-303) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
BEP20Token.constructor() (april.sol#355-363) uses literals with too many digits:
        - _totalSupply = 125000000000000000000000000 (april.sol#359)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (april.sol#320-323)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (april.sol#329-331)
increaseAllowance(address,uint256) should be declared external:
        - BEP20Token.increaseAllowance(address,uint256) (april.sol#469-472)
decreaseAllowance(address,uint256) should be declared external:
        - BEP20Token.decreaseAllowance(address,uint256) (april.sol#488-491)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-
be-declared-external
INFO:Slither:april.sol analyzed (5 contracts with 46 detectors), 7 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

There are three different types of issue detected by slither.

Description 1

Functions like renounceOwnership, transferOwnership, increaseAllowance, decreaseAllowance are declared as internal, whereas it should be marked as external. This issue has already been discussed in manual analysis. Reading link.

Description 2

Uses literals with too many digits:
BEP20Token.constructor() (april.sol#355-363) uses literals with too many digits:
- _totalSupply = 125000000000000000000000000 (april.sol#359)

Detailed reading link.

Description 3: Local variable shadowing
BEP20Token.allowance(address,address).owner (april.sol#423) shadows:
- Ownable.owner() (april.sol#301-303) (function)

BEP20Token._approve(address,address,uint256).owner (april.sol#528) shadows:
- Ownable.owner() (april.sol#301-303) (function)

Detailed reading link.

Manticore

Manticore is a symbolic execution tool for the analysis of smart contracts and binaries. It executes a program with symbolic inputs and explores all the possible states it can reach. It also detects crashes and other failure cases in binaries and smart contracts.
Manticore results throw the same warning which is similar to the Slither warning.

Mythx

MythX is a security analysis tool and API that performs static analysis, dynamic analysis, symbolic execution, and fuzzing on Ethereum smart contracts. MythX checks for and reports on the common security vulnerabilities in open industry-standard SWC Registry.

There are many contracts within the whole file. I have separately put them for analysis. Below are the reports generated for each contract separately.

Analysis a977b15c-5c32-4ae9-bfb6-6e95580461f1 MythX

Started	Thu Apr 29 2021 07:20:50 GMT+0000 (Coordinated Universal Time)
Finished	Thu Apr 29 2021 07:22:57 GMT+0000 (Coordinated Universal Time)
Mode	Quick
Client Tool	Remythx
Main Source File	Quill/April.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	4	0

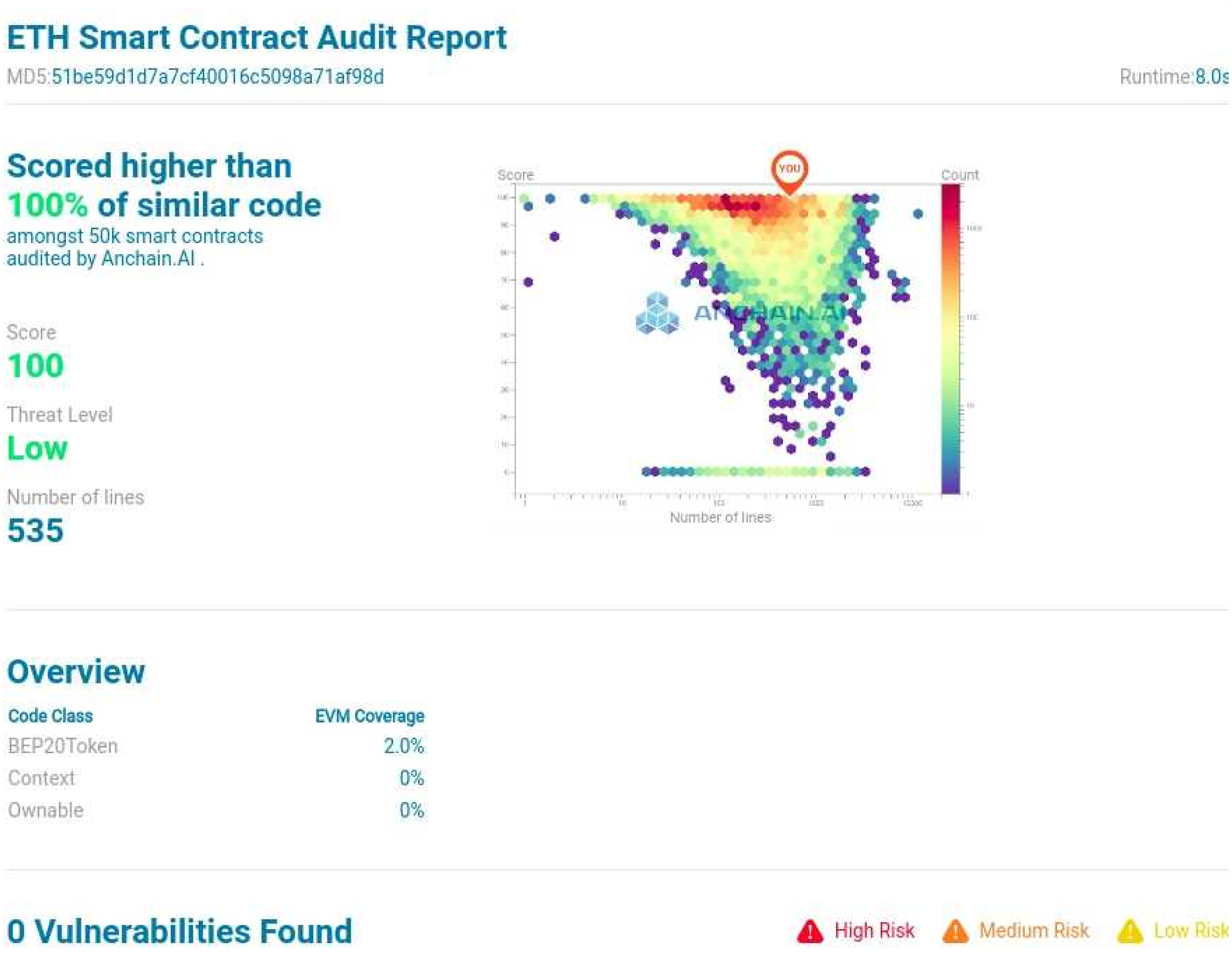
Most of the vulnerabilities generated by Mythx are discussed in the Manual sections. Mythx Report contains both Medium issues only. The pdf copy of the report can be found at this link.

Medium level vulnerability

Issue: Function could be marked as external. Already discussed in manual analysis. This issue has already been discussed in the manual audit section. anual analysis.

Anchain:

Anchain sandbox audits the security score of any Solidity-based smart contract, having analyzed the source code of every mainnet EVM smart contract plus the 1M + unique, user-uploaded smart contracts. Code has been analyzed there and got the report below.


ETH Smart Contract Audit Report

MD5:51be59d1d7a7cf40016c5098a71af98d Runtime:8.0s

Scored higher than
100% of similar code
amongst 50k smart contracts audited by Anchain.AI .

Score
100

Threat Level
Low

Number of lines
535

Overview

Code Class	EVM Coverage
BEP20Token	2.0%
Context	0%
Ownable	0%

0 Vulnerabilities Found

High Risk Medium Risk Low Risk

Vulnerability Checklist

BEP20Token

- Integer Underflow
- Integer Overflow
- Parity Multisig Bug
- Callstack Depth Attack
- Transaction-Ordering Dependency
- Timestamp Dependency
- Re-Entrancy

Context

- Integer Underflow
- Integer Overflow
- Parity Multisig Bug
- Callstack Depth Attack
- Transaction-Ordering Dependency
- Timestamp Dependency
- Re-Entrancy

Ownable

- Integer Underflow
- Integer Overflow
- Parity Multisig Bug
- Callstack Depth Attack
- Transaction-Ordering Dependency
- Timestamp Dependency
- Re-Entrancy

There were 0 vulnerabilities found by Anchain.

Suggestions

1. Missing License: Developers need to use a license according to your project. The suggestion to use here would be an SPDX license. You can find a list of licenses here: https://spdx.org/licenses/

   The SPDX License List is an integral part of the SPDX Specification. The SPDX License List itself is a list of commonly found licenses and exceptions used in free and open or collaborative software, data, hardware, or documentation. The SPDX License List includes a standardized short identifier, the full name, the license text, and a canonical permanent URL for each license and exception.

2. Follow Open Zeppelin standard for token contract making. There were few functions like Burn , Pausable and few others were missing which is usually

Disclaimer

The audit does not give any warranties on the security of the code. One audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of the code. Besides a security audit, please don't consider this report as investment advice.

Summary

The use of smart contracts is simple and the code is relatively small. Altogether the code is written and demonstrates effective use of abstraction, separation of concern, and modularity. But there are a few issues/vulnerabilities to be tackled at various security levels, it is recommended to fix them before deploying the contract on the main network. Given the subjective nature of some assessments, it will be up to the April team to decide whether any changes should be made.

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

hello@quillhash.com