



QuillAudits



Audit Report  
May, 2021



MrwebFi



# Contents

Introduction	01
Audit Goals	02
Issue Categories	03
Manual Audit	04
Automated Testing	09
Summary	13
Disclaimer	14

# Introduction

This audit report highlights the overall security of the contract given by the MrWeb Finance team. With this report, QuillHash Audit has 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, we consider the following crucial features of the code.

- Whether the implementation of ERC 20 standards.
- Whether the code is secure.
- Gas Optimization
- 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, Echidna, 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.

**Report:** All the gathered information is described in this report.

## Audit Details

**Project Name:** MRWEB

**Token symbol:** AMA

**Audit code link:** <https://tronscan.org/#/contract/TVocZFCRZ6tg8MqKCKXzZ9H2qSg29T75tK/code>

**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 in three categories.

### Security

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

### Sound Architecture

Evaluating the architect of a system through the lens of established smart contract best practice and general software practice.

### 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

	High	Medium	Low	Informational
Open	0	0	4	2



# Manual Audit

## SWC Registry test

We have tested some known SWC registry issues. Out of all tests only SWC 102 and 103. Both are low priority. We have about it above already.

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

Serial No.	Description	Comments
<u>SWC-113</u>	DoS with Failed Call	Pass: No failed call
<u>SWC-112</u>	Delegatecall to Untrusted Callee	Pass
<u>SWC-111</u>	Use of Deprecated Solidity Functions	Pass : No deprecated function used
<u>SWC-108</u>	State Variable Default Visibility	Pass: Explicitly defined visibility for all state variables
<u>SWC-107</u>	Reentrancy	Pass: Properly used
<u>SWC-106</u>	Unprotected SELF-DESTRUCT Instruction	Pass: Not found any such vulnerability
<u>SWC-104</u>	Unchecked Call Return Value	Pass: Not found any such vulnerability
<u>SWC-103</u>	Floating Pragma	Found
<u>SWC-102</u>	Outdated Compiler Version	Found: Latest version is <u>Version 0.8.4</u> . In code 0.4.0 is used
<u>SWC-101</u>	Integer Overflow and Underflow	Pass

## High level severity issues

No issues found

## Medium level severity issues

No issues found



## Low level severity issues

There were 4 low severity issues found.

### 1. Description → SWC 102: Outdated Compiler Version

```
1 // 0.5.1-c8a2
2 // Enable optimization
3 pragma solidity ^0.5.0;
4
```

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 0.8.4.

### 2. Description → SWC 103: Floating Pragma

```
1 // 0.5.1-c8a2
2 // Enable optimization
3 pragma solidity ^0.5.0;
4
```

Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively. This issue has been found in all the solidity files.

#### Remediation

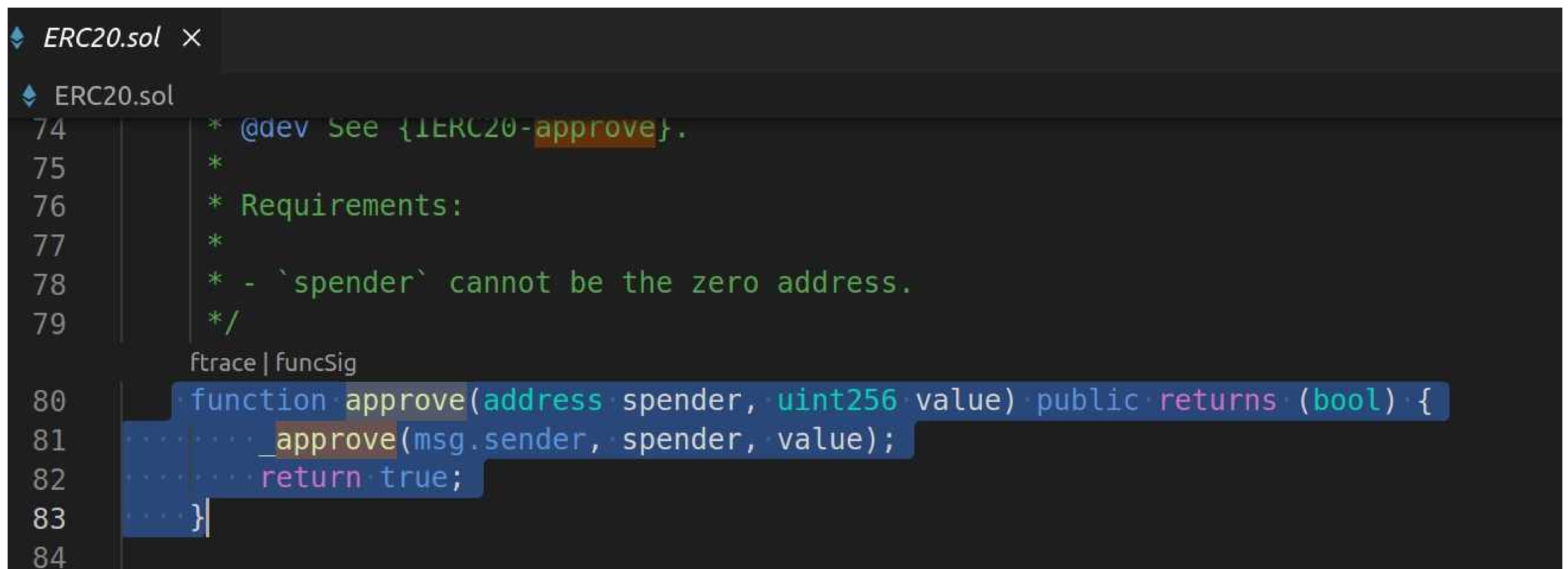
Lock the pragma version and also consider known bugs (<https://github.com/ethereum/solidity/releases>) for the compiler version that is chosen.

Pragma statements can be allowed to float when a contract is intended for consumption by other developers, as in the case with contracts in a library or EthPM package. Otherwise, the developer would need to manually update the pragma in order to compile locally.



### 3. Description: Using the approve function of the token standard [ Line 80-83 in ERC20.sol ]

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



```
ERC20.sol x
ERC20.sol
74      * @dev See {IERC20-approve}.
75      *
76      * Requirements:
77      *
78      * - `spender` cannot be the zero address.
79      */
      ftrace | funcSig
80      function approve(address spender, uint256 value) public returns (bool) {
81          _approve(msg.sender, spender, value);
82          return true;
83      }
84
```

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. Though 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](#)

### 4. Description: Prefer external to public visibility level

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 "totalSupply", "balanceOf", "transfer", "allowance", "approve", "transferFrom" in **IERC20.sol** are marked as "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark 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 Slither as well.



# 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.

```
Pragma version^0.5.0 (IERC20.sol#1) allows old versions
solc-0.5.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Docummentation#incorrect-versions-of-solidity
INFO:Detectors:
Pragma version^0.5.0 (ERC20.sol#1) allows old versions
Pragma version^0.5.0 (IERC20.sol#1) allows old versions
Pragma version^0.5.0 (SafeMath.sol#1) allows old versions
solc-0.5.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Docummentation#incorrect-versions-of-solidity
INFO:Detectors:
totalSupply() should be declared external:
- ERC20.totalSupply() (ERC20.sol#42-44)
balanceOf(address) should be declared external:
- ERC20.balanceOf(address) (ERC20.sol#49-51)
transfer(address,uint256) should be declared external:
- ERC20.transfer(address,uint256) (ERC20.sol#61-64)
allowance(address,address) should be declared external:
- ERC20.allowance(address,address) (ERC20.sol#69-71)
approve(address,uint256) should be declared external:
- ERC20.approve(address,uint256) (ERC20.sol#80-83)
transferFrom(address,address,uint256) should be declared external:
- ERC20.transferFrom(address,address,uint256) (ERC20.sol#97-101)
increaseAllowance(address,uint256) should be declared external:
- ERC20.increaseAllowance(address,uint256) (ERC20.sol#115-118)
decreaseAllowance(address,uint256) should be declared external:
- ERC20.decreaseAllowance(address,uint256) (ERC20.sol#134-137)
Reference: https://github.com/crytic/slither/wiki/Detector-Docummentation#public-function-that-could-be-declared-external
INFO:Detectors:
Pragma version^0.5.0 (SafeMath.sol#1) allows old versions
```

```
INFO:Detectors:
Pragma version^0.5.0 (SafeMath.sol#1) allows old versions
solc-0.5.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Docummentation#incorrect-versions-of-solidity
INFO:Detectors:
ERC20Detailed.constructor(string,string,uint8).name (ERC20Detailed.sol#18) shadows:
- ERC20Detailed.name() (ERC20Detailed.sol#27-29) (function)
ERC20Detailed.constructor(string,string,uint8).symbol (ERC20Detailed.sol#18) shadows:
- ERC20Detailed.symbol() (ERC20Detailed.sol#35-37) (function)
ERC20Detailed.constructor(string,string,uint8).decimals (ERC20Detailed.sol#18) shadows:
- ERC20Detailed.decimals() (ERC20Detailed.sol#51-53) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Docummentation#local-variable-shadowing
INFO:Detectors:
Pragma version^0.5.0 (ERC20.sol#1) allows old versions
Pragma version^0.5.0 (ERC20Detailed.sol#1) allows old versions
Pragma version^0.5.0 (IERC20.sol#1) allows old versions
Pragma version^0.5.0 (SafeMath.sol#1) allows old versions
Pragma version^0.5.0 (Token.sol#3) allows old versions
solc-0.5.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Docummentation#incorrect-versions-of-solidity
INFO:Detectors:
Token.constructor() (Token.sol#19-21) uses literals with too many digits:
- _mint(msg.sender,100000000 * (10 ** uint256(decimals())) (Token.sol#20)
Reference: https://github.com/crytic/slither/wiki/Detector-Docummentation#too-many-digits
INFO:Detectors:
totalSupply() should be declared external:
- ERC20.totalSupply() (ERC20.sol#42-44)
balanceOf(address) should be declared external:
- ERC20.balanceOf(address) (ERC20.sol#49-51)
transfer(address,uint256) should be declared external:
```



```

transfer(address,uint256) should be declared external:
  - ERC20.transfer(address,uint256) (ERC20.sol#61-64)
allowance(address,address) should be declared external:
  - ERC20.allowance(address,address) (ERC20.sol#69-71)
approve(address,uint256) should be declared external:
  - ERC20.approve(address,uint256) (ERC20.sol#80-83)
transferFrom(address,address,uint256) should be declared external:
  - ERC20.transferFrom(address,address,uint256) (ERC20.sol#97-101)
increaseAllowance(address,uint256) should be declared external:
  - ERC20.increaseAllowance(address,uint256) (ERC20.sol#115-118)
decreaseAllowance(address,uint256) should be declared external:
  - ERC20.decreaseAllowance(address,uint256) (ERC20.sol#134-137)
name() should be declared external:
  - ERC20Detailed.name() (ERC20Detailed.sol#27-29)
symbol() should be declared external:
  - ERC20Detailed.symbol() (ERC20Detailed.sol#35-37)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Detectors:
ERC20Detailed.constructor(string,string,uint8).name (ERC20Detailed.sol#18) shadows:
  - ERC20Detailed.name() (ERC20Detailed.sol#27-29) (function)
ERC20Detailed.constructor(string,string,uint8).symbol (ERC20Detailed.sol#18) shadows:
  - ERC20Detailed.symbol() (ERC20Detailed.sol#35-37) (function)
ERC20Detailed.constructor(string,string,uint8).decimals (ERC20Detailed.sol#18) shadows:
  - ERC20Detailed.decimals() (ERC20Detailed.sol#51-53) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Pragma version^0.5.0 (ERC20Detailed.sol#1) allows old versions
Pragma version^0.5.0 (IERC20.sol#1) allows old versions
solc-0.5.0 is not recommended for deployment

```

```

external
INFO:Detectors:
ERC20Detailed.constructor(string,string,uint8).name (ERC20Detailed.sol#18) shadows:
  - ERC20Detailed.name() (ERC20Detailed.sol#27-29) (function)
ERC20Detailed.constructor(string,string,uint8).symbol (ERC20Detailed.sol#18) shadows:
  - ERC20Detailed.symbol() (ERC20Detailed.sol#35-37) (function)
ERC20Detailed.constructor(string,string,uint8).decimals (ERC20Detailed.sol#18) shadows:
  - ERC20Detailed.decimals() (ERC20Detailed.sol#51-53) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Pragma version^0.5.0 (ERC20Detailed.sol#1) allows old versions
Pragma version^0.5.0 (IERC20.sol#1) allows old versions
solc-0.5.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
name() should be declared external:
  - ERC20Detailed.name() (ERC20Detailed.sol#27-29)
symbol() should be declared external:
  - ERC20Detailed.symbol() (ERC20Detailed.sol#35-37)
decimals() should be declared external:
  - ERC20Detailed.decimals() (ERC20Detailed.sol#51-53)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:. analyzed (12 contracts with 46 detectors), 45 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration

```

Different Issues found there are described below:

1. **Description:** Pragma version^0.5.0 (all the file) allows old versions

**Recommendation** → There was an upgrade done after the initial report which reduces the warnings and errors. Below are the results.

Reference reading: [Link](#)



**2. Description:** Functions should be declared external like totalSupply, balanceOf, transfer, allowance, approve, transferFrom, increaseAllowance, decreaseAllowance

**Recommendation** → It should be declared external.

Reference reading: [Link](#)

**3. Description:** Detection of shadowing using local variables.

**Recommendation** → Rename the local variables that shadow another component.

Reference reading: [Link](#)

**4. Description:** Token.constructor()

(Token.sol#19-21) uses literals with too many digits:

- `_mint(msg.sender, 1000000000 * (10 ** uint256(decimals())))`

(Token.sol#20)

**Recommendation** → Use:

- Ether suffix,
- Time suffix, or
- The scientific notation

Reference 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.

Report 1

Report 2

Most of the vulnerabilities generated by Mythx are discussed in the Manual sections. Mythx Report contains both Medium and low Severity issues. The pdf copy of the report can be found at [Link](#)

## Informational

1. Functions like Payable, Burnable, Pusable and a few other commonly used functions of OpenZeppelin are missing. As a best practice, it's always best to include those functions as well. [Reading link](#)
2. The license is missing from all the solidity files. Solidity introduces SPDX license identifiers so developers can specify the [license 544](#) the contract uses. (e.g. OpenZeppelin Contracts use MIT license). SPDX license identifiers should be added to the top of contract files. (see example in [OpenZeppelin Contracts ERC20.sol 377](#)) The following identifier should be added to the top of your contract (example uses MIT license):

```
// SPDX-License-Identifier: MIT
```

The license should be from one of the following: <https://spdx.org/licenses/>



## Disclaimer

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



## 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 Product Owner to decide whether any changes should be made.





# MrwebFi



## QuillAudits



Canada, India, Singapore and United Kingdom



[audits.quillhash.com](https://audits.quillhash.com)



[hello@quillhash.com](mailto:hello@quillhash.com)