QuillAudits

Audit Report
November, 2020

Contents

Introduction

During the period of October 21st, 2020 to October 25th, 2020 - Quillhash Team performed a security audit for YFUNI Finance smart contracts. The code for audit was taken from following the official smart contract link:
https://etherscan.io/address/0x7ef548ee7ca0855b18cc5372bd9a34678da43658#code

Overview of YFUNI

YFUNI (YFUNI) is a decentralized financial (DeFi) platform centred on a Single Place that allows users to easily gain much-needed access. YFUNI claims powers a decentralized savings and multi-level interest generation protocol built on Ethereum and the ecosystem can better guarantee Equality, Equity and Security for users in staking crypto currency assets. YFUNI provides access such as Staking, Farming, Vault, Barter, Borrow and Mortgage which allows users to safely interact directly with our ecosystem.

YFUNI Token Profile
Ticker: YFUNI
Decimal: 18
Token Based: ERC-20

Token Allocation
Total Supply: 30,000 YFUNI (100%)
Presale on LID: 9,000 YFUNI (30%)
Development: 10,500 YFUNI (35%)
Stake Pool: 3,000 YFUNI (10%)
Marketing: 3,000 YFUNI (10%)
Team: 2,700 YFUNI (9%)
Liquidity: 1,500 YFUNI (5%)
Lid Team: 300 YFUNI (1%)

Scope of Audit

The scope of this audit was to analyze and document YFUNI Finance contracts 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:

- Overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per intended behavior mentioned in 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.
SmartCheck.

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 security of smart contracts.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerability 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 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 with 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 level 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 level 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

	High	Medium	Low	Informational
Open	0	0	7	8
Closed	0	0	0	0

Issues Found – Code Review / Manual Testing

High severity issues

None.

Medium severity issues

None.

Low level severity issues

1. **Compiler version should be fixed**
   Solidity source files indicate the versions of the compiler they can be compiled with. It's recommended to lock the compiler version in code, as future compiler versions may handle certain language constructions in a way the developer did not foresee. Also, it's recommended to use the latest compiler version.

2. **Coding Style Issues**
   Coding style issues influence code readability and in some cases may lead to bugs in future. Smart Contracts have naming convention, indentation and code layout issues. It's recommended to use the Solidity Style Guide to fix all the issues. Consider following the Solidity guidelines on formatting the code and commenting for all the files. It can improve the overall code quality and readability.

3. **Order of layout**
   The order of functions as well as the rest of the code layout does not follow solidity style guide, please read following documentation links to understand the correct order:
   - https://solidity.readthedocs.io/en/v0.5.3/style-guide.html#order-of-layout
   - https://solidity.readthedocs.io/en/v0.5.3/style-guide.html#order-of-functions

4. **Give preference for 'bytes32' over 'string'**
   For constant values smaller than 32 bytes, give preference for a bytes32 type, since they are much cheaper than string types, which are dynamically sized.

5. **View functions should not modify the state**
   Using inline assembly that contains certain opcodes is considered as modifying the state. Functions that modify the state should not be declared as pure functions. Do not declare functions that change the state as a view. **Lines:** 21-30

6. **Use external function modifier instead of public**
   The public functions that are never called by contract should be declared external to save gas. List of functions that can be declared external:
   - YFUNI.owner() [#184-186]
   - YFUNI.isAdmin(address) [#188-190]
   - YFUNI.renounceOwnership() [#202-205]
   - YFUNI.transferOwnership(address) [#207-211]
   - YFUNI.promoteAdmin(address) [#213-217]
   - YFUNI.demoteAdmin(address) [#219-223]
   - YFUNI.name() [#225-227]
   - YFUNI.symbol() [#229-231]

- YFUNI.decimals() [#233-235]
- YFUNI.totalSupply() [#237-239]
- YFUNI.balanceOf(address) [#241-243]
- YFUNI.isBlackList(address) [#245-247]
- YFUNI.lockInfo(address) [#249-259]
- YFUNI.transfer(address,uint256) [#261-264]
- YFUNI.allowance(address,address) [#266-268]
- YFUNI.approve(address,uint256) [#270-273]
- YFUNI.transferFrom(address,address,uint256) [#275-279]
- YFUNI.transferAndLock(address,uint256,uint256) [#281-285]
- YFUNI.increaseAllowance(address,uint256) [#287-290]
- YFUNI.decreaseAllowance(address,uint256) [#292-295]
- YFUNI.burnTarget(address,uint256) [#297-300]
- YFUNI.lockTarget(address,uint256,uint256) [#302-305]
- YFUNI.unlockTarget(address) [#307-310]
- YFUNI.blacklistTarget(address) [#312-315]
- YFUNI.unblacklistTarget(address) [#317-320]

7. **Structs should be named using the CapWords style**
   Struct YFUNI.lockDetail [#152-155] is not in CapWords.

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.

1. **Implicit Visibility [#169-178]**
   It's a good practice to explicitly define visibility of state variables, functions, interface functions and fallback functions. The default visibility of state variables – internal; function – public; interface function - external.

2. **Delete variables that you don't need**
   In Ethereum, you get a gas refund for freeing up storage space. If you don't need a variable anymore, you should delete it using the delete keyword provided by solidity or by setting it to its default value.

3. **Boolean equality check**
   Boolean constants can be used directly. No need to be compared to true or false. **Lines:** 198, 214,220,326,384,392

4. **Low level calls**
   Low-level calls do not check for code existence or call success. Use of low-level calls should be avoided as they are error-prone. [#40, #65]

5. **Do not use EVM assembly. Use of assembly is error-prone and should be avoided.**

6. **Use of block.timestamp should be avoided**
   It is risky to use block.timestamp. As block.timestamp can be manipulated by miners. Therefore block.timestamp is recommended to be supplemented with some other strategy in the case of high-value/risk applications

7. It is recommended to use the library SafeERC20. SafeERC20 is a wrapper around the ERC20 interface that eliminates the need to handle boolean return values.

8. **Approve function of ERC-20 is vulnerable**
   The Multiple Withdrawal Attack is a known attack on ERC20 that allows and approves spender to transfer more than allowed by another user. More details can be found on the below link:
   https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/edit#heading=h.q96g0auxmi6a

   The above document explains in detail on how to fix the issue using three different approaches:
   a. Atomic "Compare And Set" Approve Method
   b. Separate Log Message for "TransferFrom" Transfers
   c. Four-Args Approval Event

Automated Testing

Slither

Slither is an open-source Solidity static analysis framework. This tool provides rich information about Ethereum smart contracts and has the critical properties. It 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.

- Detects vulnerable Solidity code with low false positives
- Identifies where the error condition occurs in the source code
- Easily integrates into continuous integration and Truffle builds
- Built-in 'printers' quickly report crucial contract information
- Detector API to write custom analyses in Python
- Ability to analyze contracts written with Solidity >= 0.4
- Intermediate representation (SlithIR) enables simple, high-precision analyses
- Correctly parses 99.9% of all public Solidity code
- Average execution time of less than 1 second per contract

```
INFO:Detectors:
YFUNI.constructor(string,string,uint256).name (yfuni.sol#173) shadows:
        - YFUNI.name() (yfuni.sol#225-227) (function)
YFUNI.constructor(string,string,uint256).symbol (yfuni.sol#173) shadows:
        - YFUNI.symbol() (yfuni.sol#229-231) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
YFUNI.lockInfo(address) (yfuni.sol#249-259) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp > sys.lockUntil (yfuni.sol#251)
YFUNI._transfer(address,address,uint256) (yfuni.sol#322-346) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp > sys.lockUntil (yfuni.sol#330)
YFUNI._wantLock(address,uint256,uint256) (yfuni.sol#358-371) uses timestamp for comparisons
        Dangerous comparisons:
        - sys.lockUntil > 0 && block.timestamp > sys.lockUntil (yfuni.sol#363)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (yfuni.sol#25-34) uses assembly
        - INLINE ASM (yfuni.sol#32)
Address._functionCallWithValue(address,bytes,uint256,string) (yfuni.sol#61-82) uses assembly
        - INLINE ASM (yfuni.sol#74-77)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
YFUNI.promoteAdmin(address) (yfuni.sol#213-217) compares to a boolean constant:
        -require(bool,string)(_isAdmin[newAdmin] == false,Ownable: address is already admin) (yfuni.sol#214)
YFUNI.demoteAdmin(address) (yfuni.sol#219-223) compares to a boolean constant:
        -require(bool,string)(_isAdmin[oldAdmin] == true,Ownable: address is not admin) (yfuni.sol#220)
YFUNI._transfer(address,address,uint256) (yfuni.sol#322-346) compares to a boolean constant:
        -require(bool,string)(_blacklist[sender] == false,ERC20: sender address blacklisted) (yfuni.sol#326)
YFUNI._wantblacklist(address) (yfuni.sol#382-388) compares to a boolean constant:
        -require(bool,string)(_blacklist[account] == false,ERC20: Address already in blacklist) (yfuni.sol#384)
YFUNI._wantunblacklist(address) (yfuni.sol#390-396) compares to a boolean constant:
        -require(bool,string)(_blacklist[account] == true,ERC20: Address not blacklisted) (yfuni.sol#392)
YFUNI.onlyAdmin() (yfuni.sol#197-200) compares to a boolean constant:
        -require(bool,string)(_isAdmin[_msgSender()] == true,Ownable: caller is not the administrator) (yfuni.sol#198)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Pragma version^0.7.0 (yfuni.sol#9) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
solc-0.7.1 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
```

```
        - YFUNI.name() (yfuni.sol#225-227)
symbol() should be declared external:
        - YFUNI.symbol() (yfuni.sol#229-231)
decimals() should be declared external:
        - YFUNI.decimals() (yfuni.sol#233-235)
totalSupply() should be declared external:
        - YFUNI.totalSupply() (yfuni.sol#237-239)
balanceOf(address) should be declared external:
        - YFUNI.balanceOf(address) (yfuni.sol#241-243)
isBlackList(address) should be declared external:
        - YFUNI.isBlackList(address) (yfuni.sol#245-247)
lockInfo(address) should be declared external:
        - YFUNI.lockInfo(address) (yfuni.sol#249-259)
transfer(address,uint256) should be declared external:
        - YFUNI.transfer(address,uint256) (yfuni.sol#261-264)
allowance(address,address) should be declared external:
        - YFUNI.allowance(address,address) (yfuni.sol#266-268)
approve(address,uint256) should be declared external:
        - YFUNI.approve(address,uint256) (yfuni.sol#270-273)
transferFrom(address,address,uint256) should be declared external:
        - YFUNI.transferFrom(address,address,uint256) (yfuni.sol#275-279)
transferAndLock(address,uint256,uint256) should be declared external:
        - YFUNI.transferAndLock(address,uint256,uint256) (yfuni.sol#281-285)
increaseAllowance(address,uint256) should be declared external:
        - YFUNI.increaseAllowance(address,uint256) (yfuni.sol#287-290)
decreaseAllowance(address,uint256) should be declared external:
        - YFUNI.decreaseAllowance(address,uint256) (yfuni.sol#292-295)
burnTarget(address,uint256) should be declared external:
        - YFUNI.burnTarget(address,uint256) (yfuni.sol#297-300)
lockTarget(address,uint256,uint256) should be declared external:
        - YFUNI.lockTarget(address,uint256,uint256) (yfuni.sol#302-305)
unlockTarget(address) should be declared external:
        - YFUNI.unlockTarget(address) (yfuni.sol#307-310)
blacklistTarget(address) should be declared external:
        - YFUNI.blacklistTarget(address) (yfuni.sol#312-315)
unblacklistTarget(address) should be declared external:
        - YFUNI.unblacklistTarget(address) (yfuni.sol#317-320)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:yfuni.sol analyzed (5 contracts with 46 detectors), 43 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

```
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (yfuni.sol#36-42):
        - (success) = recipient.call{value: amount}() (yfuni.sol#40)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (yfuni.sol#61-82):
        - (success,returndata) = target.call{value: weiValue}(data) (yfuni.sol#65)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Struct YFUNI.lockDetail (yfuni.sol#152-155) is not in CapWords
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformity-to-solidity-naming-conventions
INFO:Detectors:
owner() should be declared external:
        - YFUNI.owner() (yfuni.sol#184-186)
isAdmin(address) should be declared external:
        - YFUNI.isAdmin(address) (yfuni.sol#188-190)
renounceOwnership() should be declared external:
        - YFUNI.renounceOwnership() (yfuni.sol#202-205)
transferOwnership(address) should be declared external:
        - YFUNI.transferOwnership(address) (yfuni.sol#207-211)
promoteAdmin(address) should be declared external:
        - YFUNI.promoteAdmin(address) (yfuni.sol#213-217)
demoteAdmin(address) should be declared external:
        - YFUNI.demoteAdmin(address) (yfuni.sol#219-223)
name() should be declared external:
        - YFUNI.name() (yfuni.sol#225-227)
symbol() should be declared external:
        - YFUNI.symbol() (yfuni.sol#229-231)
decimals() should be declared external:
        - YFUNI.decimals() (yfuni.sol#233-235)
totalSupply() should be declared external:
        - YFUNI.totalSupply() (yfuni.sol#237-239)
balanceOf(address) should be declared external:
        - YFUNI.balanceOf(address) (yfuni.sol#241-243)
isBlackList(address) should be declared external:
        - YFUNI.isBlackList(address) (yfuni.sol#245-247)
lockInfo(address) should be declared external:
        - YFUNI.lockInfo(address) (yfuni.sol#249-259)
transfer(address,uint256) should be declared external:
        - YFUNI.transfer(address,uint256) (yfuni.sol#261-264)
allowance(address,address) should be declared external:
        - YFUNI.allowance(address,address) (yfuni.sol#266-268)
approve(address,uint256) should be declared external:
        - YFUNI.approve(address,uint256) (yfuni.sol#270-273)
```

Slither didn't raise any critical issue with YFUNI contracts. The contract was well tested and all the minor issues that were raised have been documented in the report. All other vulnerabilities of importance have already been covered in the manual audit section of the report.

SmartCheck

SmartCheck is a tool for automated static analysis of Solidity source code for security vulnerabilities and best practices. SmartCheck translates Solidity source code into an XML-based intermediate representation and checks i t against XPath patterns. SmartCheck shows significant improvements over existing alternatives in terms of false discovery rate (FDR) and false negative rate (FNR). It gave the following result for the YFUNI contract:

https://tool.smartdec.net/scan/48d5abcd79b9441188c0a144ffdd5586

SmartCheck did not detect any high severity issue. All the considerable issues raised by SmartCheck are already covered in the code review section of this report.

Mythril

Mythril is a security analysis tool for EVM bytecode. It detects security vulnerabilities in smart contracts built for Ethereum. It uses symbolic execution, SMT solving and taint analysis to detect a variety of security vulnerabilities.

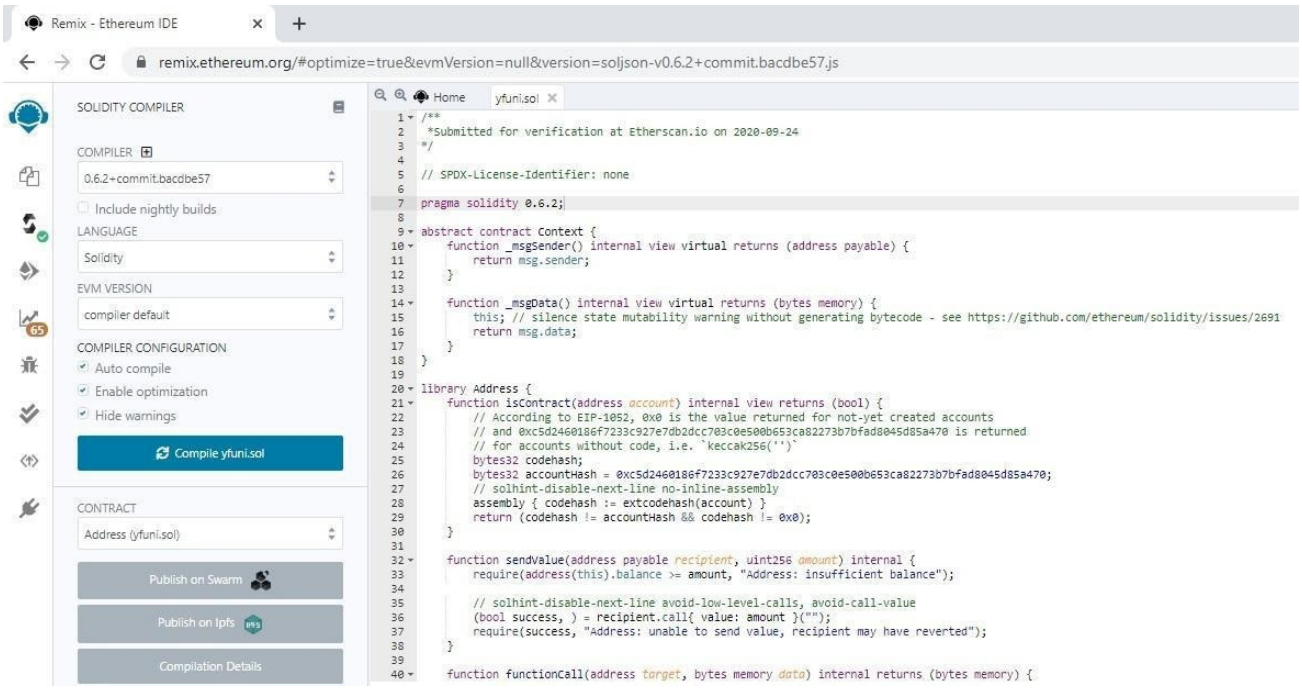
Mythril did not detect any high severity issue. All the considerable issues raised by Mythril are already covered in the code review section of this report.

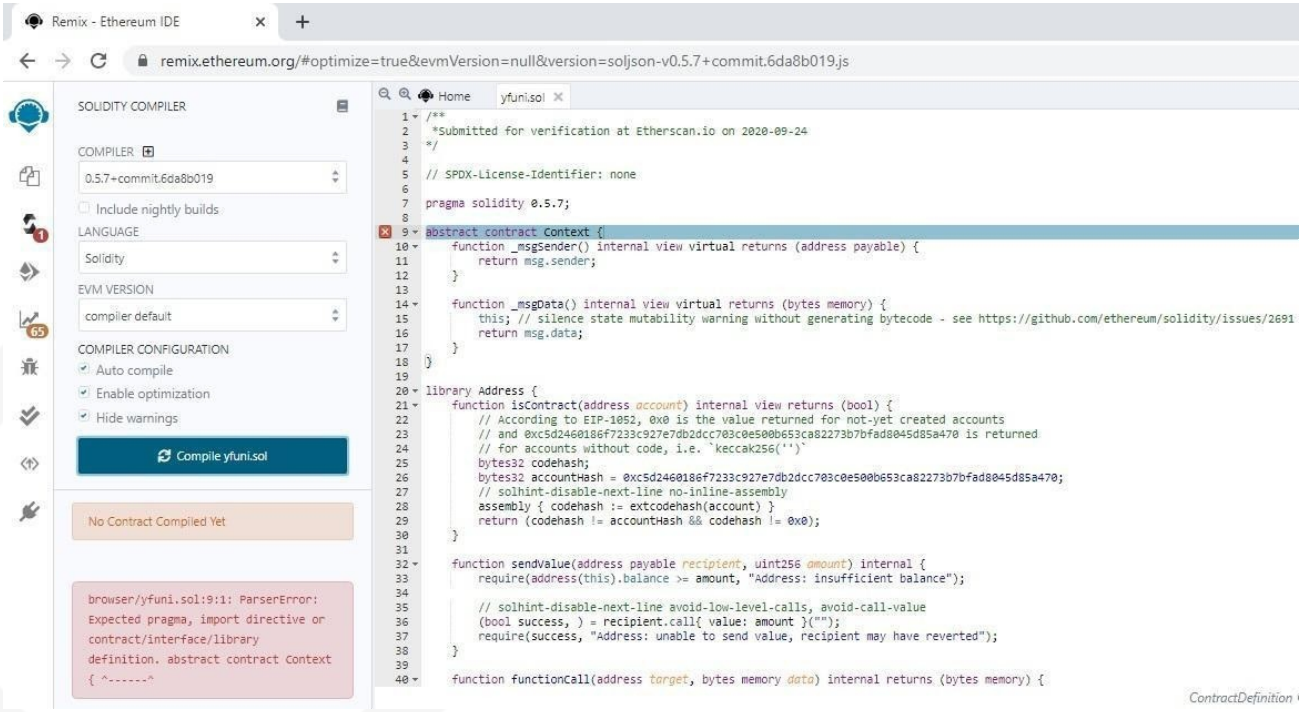
Remix IDE

Remix is a powerful, open source tool that helps you write Solidity contracts straight from the browser. Written in JavaScript, Remix supports both usage in the browser and locally.

The YFUNI smart contract was tested for various compiler versions. The smart contract compiled without any error on explicitly setting compiler version 0.6.2 and higher. It threw some warnings and these have been reported in the code review or manual testing section of this report. There was no critical or major issue reported by Remix.

It is recommended to use compiler versions 0.6.2 and higher.





Closing Summary

Overall, the smart contracts are very well written and adhere to ERC-20 guidelines. Several issues of low severity and one medium severity were found during the audit. There were no critical or major issues found that can break the intended behaviour.

Disclaimer

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

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

hello@quillhash.com