QuillAudits

Audit Report
March, 2021

Contents

Introduction

During the period of March 25th, 2021 to March 30th, 2021 – QuillHash Team performed security audit for YFUNI uEarn smart contracts. The code for audit was taken from the following link:
https://bscscan.com/address/0xbec82f72fbe9e502ec04001f7231fa1679260333#code

Updated Smart Contracts:
https://bscscan.com/address/0xc48ca15F3d2B11eccb2f9386b28cF708737BBf3a

Overview of YFUNI uEarn

YFUNI is a Yearn Inspired, next-gen suite of products for revolutionizes the Decentralized Finance (DeFi) Infrastructure. YFUNI Finance aims to provide new-innovation yield farming aggregation, lending aggregation, crypto banking investment, all on the Binance Smart Chain. We have designed it to allows users to access the best yields available in YFUNI

Features YFUNI ecosystem

uEarn
Deposit your crypto assets and get the optimal interest rate with the highest yield.
uVault
uVault is where you will earn interest when you deposit YFUNI or any stablecoin. Your deposited assets will grow and generate passive income without any risks this is similar to staking but with flexible/ varied apr
uBarter
An easy way to swap/exchange crypto assets in a P2P (Peer to Peer) way.
uBorrow
A loan system that NO requires KYC and competitive costs.
uTokens
Are built to an autonomous and non-custodial pegged asset layer for the YFUNI Decentralised Finance (DeFi) ecosystem.

uMortgage
A YFUNI pawn system that provides more benefits for customers, with up to a maximum of 75% borrowing of the total tokens is available, and all YFUNI pawned will go into staking.

Details: https://yfuni.finance/

Scope of Audit

The scope of this audit was to analyse YFUNIuEarn smart contracts codebase for quality, security, and correctness.

Check Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with (Unexpected) revert
- 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
- Address hardcoded
- Using delete for arrays
- Integer overflow/underflow
- Locked money
- Private modifier
- Revert/require functions
- Using var
- Visibility
- Using blockhash
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of Passive Income Bot smart contracts care was taken to ensure:

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

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	12	3
Closed	0	2	2	0

Issues Found – Code Review / Manual Testing

High severity issues

None.

Medium severity issues

1. **Use of incorrect OpenZeppelin contracts**
   Binance Smart Chain uses BEP-20 token standard to issue and implement tokens. It is recommended to use interface IBEP20 instead of IERC20.
   - https://academy.binance.com/en/glossary/bep-20
   - https://github.com/binance-chain/BEPs/blob/master/BEP20.md
   - https://github.com/binance-chain/bsc-genesis-contract/blob/master/contracts/bep20_template/BEP20Token.template

   Auditor Remarks: Fixed

2. **Reentrancy**
   One of the major dangers of calling external contracts is that they can take over the control flow, and make changes to your data that the calling function wasn't expecting. It's recommended that the calls to external functions/events should happen after any changes to state variables in your contract so your contract is not vulnerable to a reentrancy exploit. When control is transferred to recipient, care must be taken to not create reentrancy vulnerabilities. OpenZeppelin has its own mutex implementation you can use called ReentrancyGuard. This library provides a modifier you can apply to any function called nonReentrant that guards the function with a mutex.
   Consider using ReentrancyGuard or the checks-effects-interactions pattern.

   Following link explains more about Reentrancy and ReentrancyGuard
   - https://docs.openzeppelin.com/contracts/2.x/api/utils#ReentrancyGuard
   - https://blog.openzeppelin.com/reentrancy-after-istanbul/

Code Lines:

-Reentrancy in uEarn.sol

- claimDevBalance(address) [#132-145]
- claimReward() [#155-184]
- unstakeNow() [#227-286]
- stakeNow(uint256,uint256,uint256) [#186-225]

-Reentrancy in YFUNIuEarn.sol

- createNewEvent(uint256,uint256,uint256,uint256,uint256) [#50-88]

Auditor Remarks: Fixed

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 latest compiler version.

   Code Lines:

   -uEarn.sol [#3]

   -YFUNIuEarn.sol [#3]

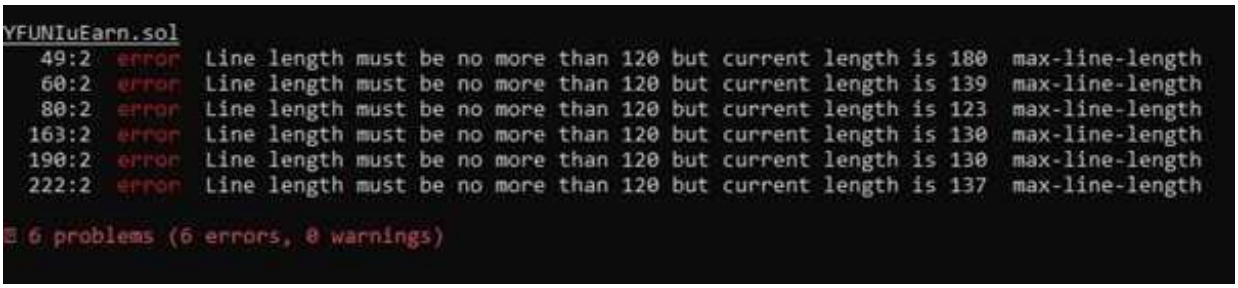
   Auditor Remarks: Not Fixed

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 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.

```
uEarn.sol
   69:2  error  Line length must be no more than 120 but current length is 201  max-line-length
   90:2  error  Line length must be no more than 120 but current length is 215  max-line-length
  102:2  error  Line length must be no more than 120 but current length is 207  max-line-length
  195:2  error  Line length must be no more than 120 but current length is 177  max-line-length
  202:2  error  Line length must be no more than 120 but current length is 127  max-line-length
  221:2  error  Line length must be no more than 120 but current length is 140  max-line-length
  305:2  error  Line length must be no more than 120 but current length is 129  max-line-length
  318:2  error  Line length must be no more than 120 but current length is 136  max-line-length
  348:2  error  Line length must be no more than 120 but current length is 140  max-line-length

✖ 9 problems (9 errors, 0 warnings)
```



Auditor Remarks: Not Fixed

3. **Order of layout**

   The order of functions as well as the rest of the code layout does not follow solidity style guide.

   Layout contract elements in the following order:

   - Pragma statements
   - Import statements
   - Interfaces
   - Libraries
   - Contracts

   Inside each contract, library or interface, use the following order:

   - Type declarations
   - State variables
   - Events
   - Functions

   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

   Auditor Remarks: Not Fixed

4. **Naming convention issues – Variables, Structs, Functions**

   It is recommended to follow all solidity naming conventions to maintain readability of code.

   Details: https://docs.soliditylang.org/en/v0.4.25/style-guide.html#naming-conventions

   Auditor Remarks: Not Fixed

5. **Be explicit about which `uint` the code is using**
   `uint` is an alias for `uint256`, but using the full form is preferable. Be consistent and use one of the forms.
   Code Lines:
   uEarn.sol
   [#13, 14, 15, 16, 18, 23, 39, 40, 50, 113, 127, 141, 153, 156, 157, 180, 183, 184]
   YFUNIuEarn.sol
   [#16, 17, 49, 52, 71, 91, 103, 117, 128, 186, 306, 319, 341, 397, 426, 439, 445]

   Auditor Remarks: Fixed

6. **Implicit Visibility**
   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.

   Auditor Remarks: Not Fixed

7. **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.

   Auditor Remarks: Not Fixed

8. **Consider using struct instead of multiple return values.**
   Replace multiple return values with a struct. It helps improve readability.
   Code Lines:
   -uEarn.sol [#308, 321, 358, 373, 459, 475]
   -YFUNIuEarn.sol [#116]

   Auditor Remarks: Not Fixed

9. **Use of different compiler versions of Solidity**
   Different compiler versions of solidity are in use - Version used: ['>=0.5.0<0.9.0', '>=0.6.0<0.9.0']
   - >=0.5.0<0.9.0 (Address.sol#3)
   - >=0.5.0<0.9.0 (Context.sol#3)
   - >=0.5.0<0.9.0 (IERC20.sol#3)
   - >=0.6.0<0.9.0 (Ownable.sol#3)
   - >=0.5.0<0.9.0 (SafeERC20.sol#3)
   - >=0.5.0<0.9.0 (SafeMath.sol#3)
   - >=0.5.0<0.9.0 (YFUNIuEarn.sol#3)
   - >=0.5.0<0.9.0 (uEarn.sol#3)

   It is recommended to a single fixed compiler version in all files including the local OpenZeppelin contracts.

   Auditor Remarks: Not Fixed

10. **Uninitialized local variable**
    It is a good practice to initialize all the variables. If a variable is meant to be initialized to zero, explicitly set it to zero.
    Code Lines:
    -YFUNIuEarn.sol [#166]

    Auditor Remarks: Not Fixed

11. **Boolean equality check**
    Boolean constants can be used directly. No need to be compare to true or false.
    Code Lines:
    -uEarn.sol [#233, 353, 470]
    -YFUNIuEarn.sol [#100, 144, 145, 199, 223]

    Auditor Remarks: Not Fixed

12. **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:

uEarn.sol
-addTokenPair(address,address,string,string,uint256,uint8,uint8) [#70-89]
-editTokenPair(uint256,address,address,string,string,uint256,uint8,uint8) [#91-101]
-editTokenPairOption(uint256,uint256,uint256,uint256,uint256,uint256, uint256,uint256,uint256) [#103-114]
-addPeriod(uint256) [#116-126]
-editPeriod(uint256,uint256) [#128-130]
-claimDevBalance(address) [#132-145]
-claimPoolToDev(address) [#147-153]
-claimReward() [#155-184]
-stakeNow(uint256,uint256,uint256) [#186-225]
-unstakeNow() [#227-286]
-getDevBalance(address) [#288-290]
-getPairInfo(uint256) [#306-317]
-getPairOptionInfo(uint256) [#319-331]
-getPairList() [#333-335]
-getPeriodList() [#337-339]
-getPeriodDetail(uint256) [#341-343]
-getRewardAllocation(address) [#345-347]
-getUserInfo(address) [#349-369]
-getUserSavedPairinfo(address) [#371-379]
-getRewardObtained(address) [#411-437]

YFUNIuEarn.sol
-createNewEvent(uint256,uint256,uint256,uint256,uint256) [#50-88]
-closeActiveEvent() [#90-95]
-userEventClaim() [#97-111]
-viewDetailEvent(uint256) [#113-125]
-viewReaminingReward(uint256) [#127-131]
-viewActiveEvent() [#133-135]

Auditor Remarks: Fixed

Informational

1. **Use of block.timestamp should be avoided**
   Malicious miners can alter the timestamp of their blocks, especially if they can gain advantages by doing so. Contracts often need access to the current timestamp to trigger time-dependent events. As Ethereum is decentralized, nodes can synchronize time only to some degree. 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.
   Code Lines:
   -uEarn.sol [#161, 164, 192, 232, 247, 248, 390, 419, 442]
   -YFUNIuEarn.sol [#219]
   Remediations:
   - https://consensys.github.io/smart-contract-best-practices/recommendations/#timestamp-dependence
   - https://consensys.github.io/smart-contract-best-practices/recommendations/#avoid-using-blocknumber-as-a-timestamp

2. **Gas optimization tips**
   As the gas costs are increasing it is highly recommended to implement gas optimization techniques to reduce gas consumption.
   https://blog.polymath.network/solidity-tips-and-tricks-to-save-gas-and-reduce-bytecode-size-c44580b218e6

3. **Use require for error handling while using transfer**
   The BEP-20/ERC-20 transfer function returns a Boolean value if the value was transferred successfully or not. It's recommended to use require for error handling when transfer function is called.

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.

```
INFO:Printers:
Compiled with solc
Number of lines: 784 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 8 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 11
Number of informational issues: 34
Number of low issues: 9
Number of medium issues: 1
Number of high issues: 3
ERCs: ERC20

+-----------+-------------+-------+--------------------+--------------+--------------------+
|    Name   | # functions |  ERCS |     ERC20 info     | Complex code |      Features      |
+-----------+-------------+-------+--------------------+--------------+--------------------+
|  Address  |      11     |       |                    |      No      |      Send ETH      |
|           |             |       |                    |              |    Delegatecall    |
|           |             |       |                    |              |      Assembly      |
|   IBEP20  |      10     | ERC20 |     No Minting     |      No      |                    |
|           |             |       | Approve Race Cond. |              |                    |
| SafeBEP20 |      6      |       |                    |      No      | Tokens interaction |
|  SafeMath |      13     |       |                    |      No      |                    |
|    uEarn  |      35     |       |                    |      No      |    Receive ETH     |
|           |             |       |                    |              |      Send ETH      |
|           |             |       |                    |              | Tokens interaction |
+-----------+-------------+-------+--------------------+--------------+--------------------+
INFO:Slither:uEarn.sol analyzed (8 contracts)
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

```
INFO:Printers:
Compiled with solc
Number of lines: 1014 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 9 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 13
Number of informational issues: 43
Number of low issues: 12
Number of medium issues: 3
Number of high issues: 3
ERCs: ERC20

+------------+-------------+-------+--------------------+--------------+--------------------+
|    Name    | # functions | ERCS  |     ERC20 info     | Complex code |      Features      |
+------------+-------------+-------+--------------------+--------------+--------------------+
|  Address   |      11     |       |                    |      No      |      Send ETH      |
|            |             |       |                    |              |    Delegatecall    |
|            |             |       |                    |              |      Assembly      |
|   IBEP20   |      10     | ERC20 |     No Minting     |      No      |                    |
|            |             |       | Approve Race Cond. |              |                    |
| SafeBEP20  |      6      |       |                    |      No      | Tokens interaction |
|  SafeMath  |      13     |       |                    |      No      |                    |
| YFUNIuEarn |      46     |       |                    |     Yes      |    Receive ETH     |
|            |             |       |                    |              |      Send ETH      |
|            |             |       |                    |              | Tokens interaction |
+------------+-------------+-------+--------------------+--------------+--------------------+
INFO:Slither:YFUNIuEarn.sol analyzed (9 contracts)
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Slither didn't raise any critical issue with smart contracts. The smart contracts were well tested and all the minor issues that were raised have been documented in the report. Also, 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 it against XPath patterns. SmartCheck shows significant improvements over existing alternatives in terms of false discovery rate (FDR) and false negative rate (FNR).

uEarn.sol

```
ruleId: SOLIDITY_VISIBILITY
patternId: b51ce0
severity: 1
line: 141
column: 12
content: IERC20(target).

ruleId: SOLIDITY_VISIBILITY
patternId: b51ce0
severity: 1
line: 141
column: 27
content: safeTransfer(_admin,

ruleId: SOLIDITY_VISIBILITY
patternId: b51ce0
severity: 1
line: 141
column: 48
content: forAdmin);

SOLIDITY_VISIBILITY :7
SOLIDITY_SAFEMATH :1
SOLIDITY_PRIVATE_MODIFIER_DONT_HIDE_DATA :9
```

YFUNIuEarn.sol

```
uleId: SOLIDITY_VISIBILITY
atternId: b51ce0
everity: 1
ine: 109
olumn: 42
ontent: ()).

uleId: SOLIDITY_VISIBILITY
atternId: b51ce0
everity: 1
ine: 109
olumn: 46
ontent: safeTransfer(msgSender,

uleId: SOLIDITY_VISIBILITY
atternId: b51ce0
everity: 1
ine: 109
olumn: 70
ontent: getUserAllocationReward);

OLIDITY_VISIBILITY :5
OLIDITY_SAFEMATH :1
OLIDITY_PRIVATE_MODIFIER_DONT_HIDE_DATA :6
```

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 issues found 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 YFUNIuEarn smart contracts were tested for various compiler versions. The smart contract compiled without any error on explicitly setting compiler **version 0.7.0 onwards.**

It is recommended to use any fixed compiler versions from 0.7.0.
The contract failed to compile from version less than 0.7.0 and 0.7.1 onwards as few functionalities in solidity have been deprecated.

The contract was successfully deployed on Rinkbey Network (0xB31544957bd23D2089A0b8CDff96a3d91D8cEe64). The gas consumption was found to be normal.

Closing Summary

Overall, the smart contracts are adhered to BEP-20 guidelines. Several issues of medium and low severity were found during the audit. There were no critical or major issues found that can break the intended behaviour. All the medium and few low priority issues were fixed by the team.

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