QuillAudits

Audit Report
March, 2021

Passive Income Bot

Contents

Introduction

During the period of March 27th, 2021 to March 28th, 2021 – QuillHash Team performed security audit for Passive Income Bot smart contracts.
The code for audit was taken from the following link:
https://etherscan.io/address/0x0a396c3ad7d2ff3556dedf39aba950de3fe54ce3
Updated Contract:
https://drive.google.com/file/d/10thicEXxuNAo3WQL_BUPXDPHJp-T_GOV/view?usp=sharing

Overview of Passive Income Bot

Passive Income Bot is an AI and Algorithm based trading bot, Dividend/ Profit sharing model for holders and early believer's and wide range of tools to help you navigate through the DEFI and Conventional Markets. The PIB ecosystem is an amalgamation of state of art crypto tools and in house-built trading strategies which can be used by PIB holders. PIB

Features Passive Income ecosystem

- Next Gen AI based algorithms and Bot
- 24/7 Automated trading
- Consistent Growth
- Passive Income.
- Offer trading tool for uniswap, pancake swap.
- Uniswap bot with features
  -Limit orders
  -Approve token before listing
  -Instant buy and sell
  -Pancake swap bot is being developed

As per the Tokenomics – 26% tokens are allocated for uniswap liquidity **(1900000 tokens)**
Details: http://pib.tools/

Scope of Audit

The scope of this audit was to analyse AlgoVest Staking smart contract 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	0	0
Closed	0	0	4	1

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. 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.
   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.

   Status: 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.

```
passiveincomebot.sol
   24:2  error  Line length must be no more than 120 but current length is 126  max-line-length
  383:2  error  Line length must be no more than 120 but current length is 130  max-line-length
  419:2  error  Line length must be no more than 120 but current length is 138  max-line-length

✖ 3 problems (3 errors, 0 warnings)
```

Status: Fixed

3. **Naming convention issues – Variables, Structs, Functions**
   Code Lines: 602-611
   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

   Status: Fixed

4. **_total should be a constant.**
   Code Lines: 604
   As per the ERC-20 token standards the total supply should be fixed. 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.

   Status: Fixed

Informational

1. **Approve function of ERC-20 is vulnerable**
   This is a recommendation for future. A security issue called “Multiple Withdrawal Attack” - originates from two methods in the ERC20 standard for approving and transferring tokens. The use of these functions in an adverse environment (e.g., front-running) could result in more tokens being spent than what was intended. This issue is still open on the GitHub and several solutions have been made to mitigate it.
   For more details on how to resolve the multiple withdrawal attack check the following document for details:

   https://drive.google.com/file/d/1skR4BpZ0VBSQICIC_eqRBgGnACf2li5X/view?usp=sharing

   Status: Fixed

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.

```
# Check Passive_Income_Bot

## Check functions
[✓] totalSupply() is present
        [✓] totalSupply() -> () (correct return value)
        [✓] totalSupply() is view
[✓] balanceOf(address) is present
        [✓] balanceOf(address) -> () (correct return value)
        [✓] balanceOf(address) is view
[✓] transfer(address,uint256) is present
        [✓] transfer(address,uint256) -> () (correct return value)
        [✓] Transfer(address,address,uint256) is emitted
[✓] transferFrom(address,address,uint256) is present
        [✓] transferFrom(address,address,uint256) -> () (correct return value)
        [✓] Transfer(address,address,uint256) is emitted
[✓] approve(address,uint256) is present
        [✓] approve(address,uint256) -> () (correct return value)
        [✓] Approval(address,address,uint256) is emitted
[✓] allowance(address,address) is present
        [✓] allowance(address,address) -> () (correct return value)
        [✓] allowance(address,address) is view
[✓] name() is present
        [✓] name() -> () (correct return value)
        [✓] name() is view
[✓] symbol() is present
        [✓] symbol() -> () (correct return value)
        [✓] symbol() is view
[✓] decimals() is present
        [✓] decimals() -> () (correct return value)
        [✓] decimals() is view

## Check events
[✓] Transfer(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed
[✓] Approval(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed
```

```
INFO:Printers:
Compiled with solc
Number of lines: 610 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 6 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 15
Number of informational issues: 5
Number of low issues: 0
Number of medium issues: 0
Number of high issues: 0

ERCs: ERC20

+--------------------+-------------+-------+--------------------+--------------+----------+
|        Name        | # functions | ERCS  |     ERC20 info     | Complex code | Features |
+--------------------+-------------+-------+--------------------+--------------+----------+
|      SafeMath      |      8      |       |                    |      No      |          |
| Passive_Income_Bot |      32     | ERC20 |     No Minting     |      No      |          |
|                    |             |       | Approve Race Cond. |              |          |
+--------------------+-------------+-------+--------------------+--------------+----------+
INFO:Slither:Passive_Income_Bot.sol analyzed (6 contracts)
```

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).

```
ruleId: SOLIDITY_PRIVATE_MODIFIER_DONT_HIDE_DATA
patternId: 5616b2
severity: 1
line: 273
column: 10
content: private

ruleId: SOLIDITY_PRIVATE_MODIFIER_DONT_HIDE_DATA
patternId: 5616b2
severity: 1
line: 551
column: 12
content: private

ruleId: SOLIDITY_PRIVATE_MODIFIER_DONT_HIDE_DATA
patternId: 5616b2
severity: 1
line: 604
column: 12
content: private

ruleId: SOLIDITY_SAFEMATH
patternId: 837cac
severity: 1
line: 263
column: 4
content: usingSafeMathforuint256;

SOLIDITY_SAFEMATH :1
SOLIDITY_PRAGMAS_VERSION :1
SOLIDITY_PRIVATE_MODIFIER_DONT_HIDE_DATA :8
SOLIDITY_ADDRESS_HARDCODED :3
SOLIDITY_ERC20_APPROVE :1
```

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.

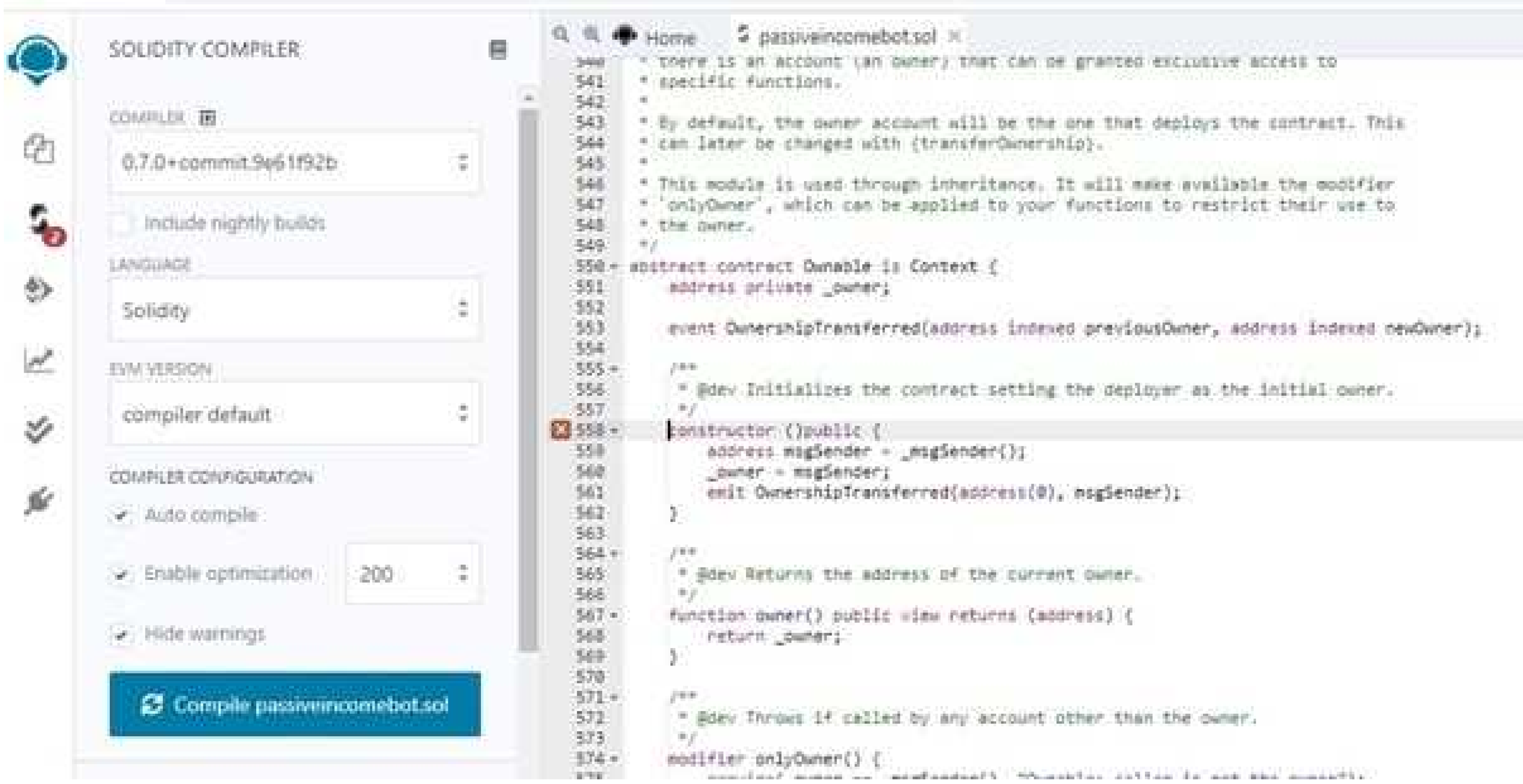
Mythril

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 Passive Income Bot smart contracts was tested for various compiler versions. The smart contract compiled without any error on explicitly setting compiler version 0.7.0 or higher.



It is recommended to use any fixed compiler versions from 0.6.0 to 0.6.12.

The contract was successfully deployed on Rinkbey Network (0x27DB1139eA9fdCAE4638cA6Ef766b00940dCc71e).
https://rinkeby.etherscan.io/tx/0xf8f170bdb422dee1a096fa849ed769ab982d01f10b4c10220e61b34bcced2ff1

The gas consumption was found to be normal.

Closing Summary

Overall, the smart contracts are adhered to ERC-20 guidelines. Several issues of low severity were found and fixed 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 Passive Income 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 Passive Income Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Passive Income Bot

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

hello@quillhash.com